

SECURITY

IKEv2 IPsec Virtual Private Networks

Understanding and Deploying IKEv2, IPsec VPNs, and FlexVPN in Cisco IOS

Graham Bartlett, CCIE No. 26709
Amjad Inamdar, CISSP No. 460898

ciscopress.com

About This E-Book

EPUB is an open, industry-standard format for e-books. However, support for EPUB and its many features varies across reading devices and applications. Use your device or app settings to customize the presentation to your liking. Settings that you can customize often include font, font size, single or double column, landscape or portrait mode, and figures that you can click or tap to enlarge. For additional information about the settings and features on your reading device or app, visit the device manufacturer's Web site.

Many titles include programming code or configuration examples. To optimize the presentation of these elements, view the e-book in single-column, landscape mode and adjust the font size to the smallest setting. In addition to presenting code and configurations in the reflowable text format, we have included images of the code that mimic the presentation found in the print book; therefore, where the reflowable format may compromise the presentation of the code listing, you will see a "Click here to view code image" link. Click the link to view the print-fidelity code image. To return to the previous page viewed, click the Back button on your device or app.

IKEv2 IPsec Virtual Private Networks

Understanding and Deploying IKEv2, IPsec VPNs, and FlexVPN in Cisco IOS

Graham Bartlett, CCIE No. 26709
Amjad Inamdar, CISSP No. 460898

Cisco Press
800 East 96th Street
Indianapolis, Indiana, 46240 USA

IKEv2 IPsec Virtual Private Networks

Understanding and Deploying IKEv2, IPsec VPNs, and FlexVPN in Cisco IOS

Graham Bartlett, CCIE No. 26709
Amjad Inamdar, CISSP No. 460898

Copyright © 2017 Cisco Systems, Inc.

Cisco Press logo is a trademark of Cisco Systems, Inc.

Published by:
Cisco Press
800 East 96th Street
Indianapolis, IN 46240 USA

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission from the publisher, except for the inclusion of brief quotations in a review.

Printed in the United States of America 1 2 3 4 5 6 7 8 9 0

First Printing August 2016

Library of Congress Cataloging-in-Publication Number: 2016947704

ISBN-13: 978-1-58714-460-8

ISBN-10: 1-58714-460-3

Warning and Disclaimer

This book is designed to provide information about IKEv2 and IPsec VPNs on Cisco IOS. Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied.

The information is provided on an “as-is” basis. The authors, Cisco Press, and Cisco Systems, Inc. shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the discs or programs that may accompany it.

The opinions expressed in this book belong to the author and are not necessarily those of Cisco Systems, Inc.

Feedback Information

At Cisco Press, our goal is to create in-depth technical books of the highest quality and value. Each book is crafted with care and precision, undergoing rigorous development that involves the unique expertise of members from the professional technical community.

Readers’ feedback is a natural continuation of this process. If you have any comments regarding how

we could improve the quality of this book, or otherwise alter it to better suit your needs, you can contact us through email at feedback@ciscopress.com. Please make sure to include the book title and ISBN in your message.

We greatly appreciate your assistance.

Trademark Acknowledgments

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Cisco Press or Cisco Systems, Inc. cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.



Americas Headquarters
Cisco Systems. Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV
Amsterdam, The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at **www.cisco.com/go/offices**.

CCDE, CCENT, Cisco Eos, Cisco HealthPresence, the Cisco logo, Cisco Lumin, Cisco Nexus, Cisco Stadium Vision, Cisco Telepresence, Cisco WebEx, DCE, and Welcome to the Human Network are trademarks; Changing the Way We Work. Live, Play, and Learn and Cisco Store are service marks; and Access Registrar, Aironet, AsyncOS. Bringing the Meeting To You. Catalyst, CCDA, CCDP, CCIE, CCIP, CCNA, CCNP, CCSP, CCVP, Cisco, the Cisco Certified Internetwork Expert logo, Cisco IOS, Cisco Press, Cisco Systems, Cisco Systems Capital, the Cisco Systems logo, Cisco Unity, Collaboration Without Limitation, EtherFast, EtherSwitch, Event Center, Fast Step, Follow Me

Browsing, FormShare, GigaDrive, HomeLink, Internet Quotient, IOS, Phone, iQuick Study, IronPort, the IronPort logo, LightStream, Linksys, MediaTone, MeetingPlace, MeetingPlace Chime Sound, MGX, Networkers, Networking Academy. Network Registrar, PCNow, PIX, PowerPanels, ProConnect, ScriptShare, SenderBase, SMARTnet, Spectrum Expert. StackWise, The Fastest Way to Increase Your Internet Quotient, TransPath, WebEx, and the WebEx logo are registered trademarks of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries.

All other trademarks mentioned in this document or website are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (0812R)

Editor in Chief: Mark Taub

Product Line Manager: Brett Bartow

Acquisitions Editor: Brett Bartow

Business Operation Manager, Cisco Press: Ronald Fligge

Managing Editor: Sandra Schroeder

Development Editor: Ellie Bru

Project Editor: Mandie Frank

Copy Editor: Lori Martinsek

Technical Editors: Alex Honore, Olivier Pelerin

Team Coordinator: Tammi Barnett

Cover Designer: Chuti Prasertsith

Composition: codeMantra

Indexer: Lisa Stumpf

About the Authors

Graham Bartlett, CCIE No. 26709, has designed a number of large scale Virtual Private Networks within the UK and worked with customers throughout the world using IKEv2 and Next Generation Encryption. Graham's interests include Security and Virtual Private Networks. Within this space he has discovered zero-day vulnerabilities, including the highest severity security advisory in the March 2015 Cisco IOS software and IOS XE software security advisory bundled publication. He has contributed to numerous IETF RFCs, and has intellectual property published as prior art. He is a CiscoLive speaker and has developed Cisco Security exam content (CCIE/CCNP). He is a CCP (Senior) IA Architect, CCP (Practitioner) Security & Information Risk Advisor, CCNP, CISSP, Cisco Security Ninja and holds a BSc(Hons) in Computer Systems and Networks.

Amjad Inamdar, CISSP No. 460898, is a Senior Technical Leader with Cisco IOS Security Engineering, India. He has primarily worked on design, development and deployment of Cisco IOS secure connectivity solutions including the industry leading FlexVPN, DMVPN, GETVPN and EzVPN solutions and is currently working on the Cisco next generation SD-WAN solution. He has contributed to IETF drafts, holds a Cisco patent and has prior art publications. He holds many industry certifications including CISSP, CCSK, CCNP Security, CCDP, CCNP R/S, CCNA (SP, Data Center, Wireless, Voice), Cisco Security Ninja and has presented security at conferences, internal forums and to Cisco customers and partners. He holds a degree (B.E) in Electronics and Communication Engineering.

Note from the Authors

Alex Honore was originally an author of this book, but due to commitments opted to become a Technical Reviewer. Alex has been a fundamental member of the team, a number of chapters were originally written by Alex in addition to some of the diagrams.

About the Technical Reviewers

Alex Honore, (CCIE Security No. 19553) has been with Cisco since 2005 and currently works as a Technical Leader in Cisco's Security Business Group, specializing in leading-edge network and threat analytics. He was a senior engineer in Cisco Technical Services for 9 years, focusing on advanced troubleshooting and escalation support for VPN, Security, and Content Networking solutions in the Technical Assistance Center (TAC), as well as consulting for Cisco Advanced Services and speaking regularly at Cisco Live on the topic of IPsec VPNs. Alex holds a M.Sc. degree of Electrical Engineering and Telecommunications from the Faculty of Engineering in Mons, Belgium.

Olivier Pelerin (CCIE Security No. 20306) has more than 16 years of experience in networking and security. He joined Cisco TAC (Cisco Technical Assistance Center) as a customer support engineer back in 2005. He is still working for TAC as a Technical Leader, focusing on escalations around VPN solutions. He has been involved in FlexVPN since the start and is co-leading the development of the packet tracer on ISR-NG/ASR1000. Olivier is a distinguished speaker at CiscoLive. He holds a degree in Geography from Univesité Catholique de Louvain (Belgium).

Dedications

Graham Bartlett:

I dedicate this book to my loving family: Lorna, Edward, Rose, and my parents.

Amjad Inamdar:

I dedicate this book to my parents, Sayedrasul and Ayesha Inamdar, who have been my role models and inspiration, and to my entire family.

Acknowledgments

Graham Bartlett:

Alex Honore deserves a very special mention. Alex originally was an author, but due to work and family commitments became a reviewer. Alex has been a diligent reviewer due to his ability to break complex topics into many simple layers. I want to thank Alex for his patience and the help he has provided. We are honored to have him as a reviewer and he is a great asset to Cisco. I know if Alex ever writes a book himself, it will be, for me, the perfect book.

Olivier Pelerin has been a fantastic reviewer and brought a lot of issues and resolutions to our attention that otherwise we would have missed. Olivier is 100% focused on customers and many of the tips, tricks, and guidance contained within need to be credited to him. Many times I've had late-night chats regarding IKEv2 and our implementation. Olivier should also be credited for improving Cisco's IKEv2 usability, stability, and serviceability, which ultimately has a positive effect on you—the customer.

Thank you both—I treat you as good friends and could not have achieved this without your knowledge and skills.

Frederic Detienne: Fred—thank you for leading the creation of DMVPN, Flex, and IKEv2 on Cisco IOS. FlexVPN was my main motivation for joining Cisco, and it's now apparent that as VPN Architect you have developed a technology that is unique within the industry. It's a great pleasure and honor to work with you.

Raffaele Brancaleoni: Raff, thanks for all the help you provided with the development of this book. You've been a great mentor and a good friend.

The many great VPN engineers in Cisco TAC: Piotr and the fantastic Krakow crew, Wen, Atri, Jay, Frank, Marcin, Istvan, Nitin, Tom. Xavier and PSIRT. All of the IKEv2 developers that have helped: Sairam, Manish, Balaji, Tapesh. Crypto team: Anthony, David, Scott and Panos.

Richard Gallagher: A great engineer and friend.

All the customers that I work with, who are not just customers of Cisco, but friends. You know who you are.

Thanks for the support of my management team, Francois, Khalid, David and Ashley.

Thanks to all my good friends and colleagues within Cisco and within the industry; you know who you are.

Also thanks for the support and patience of the Cisco Press team, namely Ellie and Brett.

Amjad Inamdar:

First, I would like to thank God for this opportunity to write a book which I have always wanted to do. Special thanks to my wife, Huzefiya, for being a constant companion in writing this book and her encouragement and support; my children, Sobia, Aleena, Arfa, and Ammar, for their patience and understanding while I spent many of the weekends and holidays writing this book. Thanks to my brother, sisters, and family members for their support.

Graham Bartlett—Thank you for launching me onto this journey by offering me the chance to be the co-author. I look forward to continuing this journey with you. Thanks for your patience and all your help, you are a great friend.

Frederic Detienne—Thank you for being my mentor. I have truly enjoyed working with you in designing and implementing the Cisco IOS IKEv2 and FlexVPN solution.

Alex Honore—Thank you for your excellent review. You have been an inspiration and a great friend and guide. I will always cherish our work together on IKEv2 and FlexVPN.

Olivier Pelerin—Thank you for your detailed review comments, most of which were customer focused. I value our work together on FlexVPN scalability.

Thanks to Raffaele Brancaleoni for his review comments and to Yeleshwarapu Sairam, Tapesh Maheshwari, Kalyani Garigipati from the IKEv2 and FlexVPN development team for their input.

Thanks to my management, colleagues, and friends at Cisco for their support.

Contents at a Glance

Foreword

Introduction

Part I Understanding IPsec VPNs

Chapter 1 Introduction to IPsec VPNs

Part II Understanding IKEv2

Chapter 2 IKEv2: The Protocol

Chapter 3 Comparison of IKEv1 and IKEv2

Part III IPsec VPNs on Cisco IOS

Chapter 4 IOS IPsec Implementation

Part IV IKEv2 Implementation

Chapter 5 IKEv2 Configuration

Chapter 6 Advanced IKEv2 Features

Chapter 7 IKEv2 Deployments

Part V FlexVPN

Chapter 8 Introduction to FlexVPN

Chapter 9 FlexVPN Server

Chapter 10 FlexVPN Client

Chapter 11 FlexVPN Load Balancer

Chapter 12 FlexVPN Deployments

Part VI IPsec VPN Maintenance

Chapter 13 Monitoring IPsec VPNs

Chapter 14 Troubleshooting IPsec VPNs

Part VII IPsec Overhead

Chapter 15 IPsec Overhead and Fragmentation

Part VIII Migration to IKEv2

Chapter 16 Migration Strategies

Index

Contents

Foreword

Introduction

Part I Understanding IPsec VPNs

Chapter 1 Introduction to IPsec VPNs

The Need and Purpose of IPsec VPNs

Building Blocks of IPsec

Security Protocols

Security Associations

Key Management Protocol

IPsec Security Services

Access Control

Anti-replay Services

Confidentiality

Connectionless Integrity

Data Origin Authentication

Traffic Flow Confidentiality

Components of IPsec

Security Parameter Index

Security Policy Database

Security Association Database

Peer Authorization Database

Lifetime

Cryptography Used in IPsec VPNs

Symmetric Cryptography

Asymmetric Cryptography

The Diffie-Hellman Exchange

Public Key Infrastructure

Public Key Cryptography

Certificate Authorities

Digital Certificates

Digital Signatures Used in IKEv2

Pre-Shared-Keys, or Shared Secret

Encryption and Authentication

IP Authentication Header
Anti-Replay
IP Encapsulating Security Payload (ESP)
Authentication
Encryption
Anti-Replay
Encapsulation Security Payload Datagram Format
Encapsulating Security Payload Version 3
Extended Sequence Numbers
Traffic Flow Confidentiality
Dummy Packets
Modes of IPsec
IPsec Transport Mode
IPsec Tunnel Mode
Summary
References

Part II Understanding IKEv2

Chapter 2 IKEv2: The Protocol
IKEv2 Overview
The IKEv2 Exchange
IKE_SA_INIT
Diffie-Hellman Key Exchange
Security Association Proposals
Security Parameter Index (SPI)
Nonce
Cookie Notification
Certificate Request
HTTP_CERT_LOOKUP_SUPPORTED
Key Material Generation
IKE_AUTH
Encrypted and Authenticated Payload
Encrypted Payload Structure
Identity
Authentication
Signature-Based Authentication
(Pre) Shared-Key-Based Authentication

EAP
Traffic Selectors
Initial Contact
CREATE_CHILD_SA
IPsec Security Association Creation
IPsec Security Association Rekey
IKEv2 Security Association Rekey
IKEv2 Packet Structure Overview
The INFORMATIONAL Exchange
Notification
Deleting Security Associations
Configuration Payload Exchange
Dead Peer Detection/Keepalive/NAT Keepalive
IKEv2 Request – Response
IKEv2 and Network Address Translation
NAT Detection
Additions to RFC7296
RFC5998 An Extension for EAP-Only Authentication in IKEv2
RFC5685 Redirect Mechanism for the Internet Key Exchange Protocol Version 2 (IKEv2)
RFC6989 Additional Diffie-Hellman Tests for the Internet Key Exchange Protocol Version 2 (IKEv2)
RFC6023 A Childless Initiation of the Internet Key Exchange Version 2 (IKEv2) Security Association (SA)
Summary
References

Chapter 3 Comparison of IKEv1 and IKEv2

Brief History of IKEv1
Exchange Modes
IKEv1
IKEv2
Anti-Denial of Service
Lifetime
Authentication
High Availability
Traffic Selectors
Use of Identities

Network Address Translation
Configuration Payload
Mobility & Multi-homing
Matching on Identity
Reliability
Cryptographic Exchange Bloat
Combined Mode Ciphers
Continuous Channel Mode
Summary
References

Part III IPsec VPNs on Cisco IOS

Chapter 4 IOS IPsec Implementation
Modes of Encapsulation
GRE Encapsulation
GRE over IPsec
IPsec Transport Mode with GRE over IPsec
IPsec Tunnel mode with GRE over IPsec
Traffic
Multicast Traffic
Non-IP Protocols
The Demise of Crypto Maps
Interface Types
Virtual Interfaces: VTI and GRE/IPsec
Traffic Selection by Routing
Static Tunnel Interfaces
Dynamic Tunnel Interfaces
sVTI and dVTI
Multipoint GRE
Tunnel Protection and Crypto Sockets
Implementation Modes
Dual Stack
Mixed Mode
Auto Tunnel Mode
VRF-Aware IPsec
VRF in Brief

VRF-Aware GRE and VRF-Aware IPsec
VRF-Aware GRE over IPsec
Summary
Reference

Part IV IKEv2 Implementation

Chapter 5 IKEv2 Configuration
IKEv2 Configuration Overview
The Guiding Principle
Scope of IKEv2 Configuration
IKEv2 Configuration Constructs
IKEv2 Proposal
Configuring the IKEv2 Proposal
Configuring IKEv2 Encryption
Configuring IKEv2 Integrity
Configuring IKEv2 Diffie-Hellman
Configuring IKEv2 Pseudorandom Function
Default IKEv2 Proposal
IKEv2 Policy
Configuring an IKEv2 Policy
Configuring IKEv2 Proposals under IKEv2 Policy
Configuring Match Statements under IKEv2 Policy
Default IKEv2 Policy
IKEv2 Policy Selection on the Initiator
IKEv2 Policy Selection on Responder
IKEv2 Policy Configuration Examples
Per-peer IKEv2 Policy
IKEv2 Policy with Multiple Proposals
IKEv2 Keyring
Configuring IKEv2 Keyring
Configuring a Peer Block in Keyring
Key Lookup on Initiator
Key Lookup on Responder
IKEv2 Keyring Configuration Example
IKEv2 Keyring Key Points
IKEv2 Profile
IKEv2 Profile as Peer Authorization Database

Configuring IKEv2 Profile
Configuring Match Statements in IKEv2 Profile
Matching any Peer Identity
Defining the Scope of IKEv2 Profile
Defining the Local IKE Identity
Defining Local and Remote Authentication Methods
IKEv2 Dead Peer Detection
IKEv2 Initial Contact
IKEv2 SA Lifetime
NAT Keepalives
IVRF (inside VRF)
Virtual-Template Interface
Disabling IKEv2 Profile
Displaying IKEv2 Profiles
IKEv2 Profile Selection on Initiator and Responder
IKEv2 Profile Key Points
IKEv2 Global Configuration
HTTP URL-based Certificate Lookup
IKEv2 Cookie Challenge
IKEv2 Call Admission Control
IKEv2 Window Size
Dead Peer Detection
NAT Keepalive
IKEv2 Diagnostics
PKI Configuration
Certificate Authority
Public-Private Key Pair
PKI Trustpoint
PKI Example
IPsec Configuration
IPsec Profile
IPsec Configuration Example
Smart Defaults
Summary

Chapter 6 Advanced IKEv2 Features
Introduction to IKEv2 Fragmentation

IP Fragmentation Overview
IKEv2 and Fragmentation
IKEv2 SGT Capability Negotiation
IKEv2 Session Authentication
IKEv2 Session Deletion on Certificate Revocation
IKEv2 Session Deletion on Certificate Expiry
IKEv2 Session Lifetime
Summary
References

Chapter 7 IKEv2 Deployments
Pre-shared-key Authentication with Smart Defaults
Elliptic Curve Digital Signature Algorithm Authentication
RSA Authentication Using HTTP URL Lookup
IKEv2 Cookie Challenge and Call Admission Control
Summary

Part V FlexVPN

Chapter 8 Introduction to FlexVPN
FlexVPN Overview
The Rationale
FlexVPN Value Proposition
FlexVPN Building Blocks
IKEv2
Cisco IOS Point-to-Point Tunnel Interfaces
Configuring Static P2P Tunnel Interfaces
Configuring Virtual-Template Interfaces
Auto-Detection of Tunnel Encapsulation and Transport
Benefits of Per-Peer P2P Tunnel Interfaces
Cisco IOS AAA Infrastructure
Configuring AAA for FlexVPN
IKEv2 Name Mangler
Configuring IKEv2 Name Mangler
Extracting Name from FQDN Identity
Extracting Name from Email Identity
Extracting Name from DN Identity
Extracting Name from EAP Identity

IKEv2 Authorization Policy
Default IKEv2 Authorization Policy
FlexVPN Authorization
Configuring FlexVPN Authorization
FlexVPN User Authorization
FlexVPN User Authorization, Using an External AAA Server
FlexVPN Group Authorization
FlexVPN Group Authorization, Using a Local AAA Database
FlexVPN Group Authorization, Using an External AAA Server
FlexVPN Implicit Authorization
FlexVPN Implicit Authorization Example
FlexVPN Authorization Types: Co-existence and Precedence
User Authorization Taking Higher Precedence
Group Authorization Taking Higher Precedence
FlexVPN Configuration Exchange
Enabling Configuration Exchange
FlexVPN Usage of Configuration Payloads
Configuration Attributes and Authorization
Configuration Exchange Examples
FlexVPN Routing
Learning Remote Subnets Locally
Learning Remote Subnets from Peer
Summary

Chapter 9 FlexVPN Server

Sequence of Events
EAP Authentication
EAP Methods
EAP Message Flow
EAP Identity
EAP Timeout
EAP Authentication Steps
Configuring EAP
EAP Configuration Example
AAA-based Pre-shared Keys
Configuring AAA-based Pre-Shared Keys
RADIUS Attributes for AAA-Based Pre-Shared Keys

AAA-Based Pre-Shared Keys Example
Accounting
Per-Session Interface
Deriving Virtual-Access Configuration from a Virtual-Template
Deriving Virtual-Access Configuration from AAA Authorization
The interface-config AAA Attribute
Deriving Virtual-Access Configuration from an Incoming Session
Virtual-Access Cloning Example
Auto Detection of Tunnel Transport and Encapsulation
RADIUS Packet of Disconnect
Configuring RADIUS Packet of Disconnect
RADIUS Packet of Disconnect Example
RADIUS Change of Authorization (CoA)
Configuring RADIUS CoA
RADIUS CoA Examples
Updating Session QoS Policy, Using CoA
Updating the Session ACL, Using CoA
IKEv2 Auto-Reconnect
Auto-Reconnect Configuration Attributes
Smart DPD
Configuring IKEv2 Auto-Reconnect
User Authentication, Using AnyConnect-EAP
AnyConnect-EAP
AnyConnect-EAP XML Messages for User Authentication
Configuring User Authentication, Using AnyConnect-EAP
AnyConnect Configuration for Aggregate Authentication
Dual-factor Authentication, Using AnyConnect-EAP
AnyConnect-EAP XML Messages for dual-factor authentication
Configuring Dual-factor Authentication, Using AnyConnect-EAP
RADIUS Attributes Supported by the FlexVPN Server
Remote Access Clients Supported by FlexVPN Server
FlexVPN Remote Access Client
Microsoft Windows7 IKEv2 Client
Cisco IKEv2 AnyConnect Client
Summary
Reference

Chapter 10 FlexVPN Client
- Introduction
- FlexVPN Client Overview
  - FlexVPN Client Building Blocks
  - *IKEv2 Configuration Exchange*
  - *Static Point-to-Point Tunnel Interface*
  - *FlexVPN Client Profile*
  - *Object Tracking*
  - *NAT*
  - FlexVPN Client Features
  - *Dual Stack Support*
  - *EAP Authentication*
  - *Dynamic Routing*
  - *Support for EzVPN Client and Network Extension Modes*
  - *Advanced Features*
- Setting up the FlexVPN Server
- EAP Authentication
- Split-DNS
  - Components of Split-DNS
- Windows Internet Naming Service (WINS)
- Domain Name
- FlexVPN Client Profile
- Backup Gateways
  - Resolution of Fully Qualified Domain Names
  - Reactivating Peers
  - Backup Gateway List
- Tunnel Interface
  - Tunnel Source
  - Tunnel Destination
- Tunnel Initiation
  - Automatic Mode
  - Manual Mode
  - Track Mode
  - *Tracking a List of Objects, Using a Boolean Expression*
- Dial Backup
- Backup Group

Network Address Translation
Design Considerations
Use of Public Key Infrastructure and Pre-Shared Keys
The Power of Tracking
Tracked Object Based on Embedded Event Manager
Troubleshooting FlexVPN Client
Useful Show Commands
Debugging FlexVPN Client
Clearing IKEv2 FlexVPN Client Sessions
Summary

Chapter 11 FlexVPN Load Balancer
Introduction
Components of the FlexVPN Load Balancer
IKEv2 Redirect
Hot Standby Routing Protocol
FlexVPN IKEv2 Load Balancer
Cluster Load
IKEv2 Redirect
Redirect Loops
FlexVPN Client
Troubleshooting IKEv2 Load Balancing
IKEv2 Load Balancer Example
Summary

Chapter 12 FlexVPN Deployments
Introduction
FlexVPN AAA-Based Pre-Shared Keys
Configuration on the Branch-1 Router
Configuration on the Branch-2 Router
Configuration on the Hub Router
Configuration on the RADIUS Server
FlexVPN User and Group Authorization
FlexVPN Client Configuration at Branch 1
FlexVPN Client Configuration at Branch 2
Configuration on the FlexVPN Server
Configuration on the RADIUS Server

Logs Specific to FlexVPN Client-1
Logs Specific to FlexVPN Client-2
FlexVPN Routing, Dual Stack, and Tunnel Mode Auto
FlexVPN Spoke Configuration at Branch-1
FlexVPN Spoke Configuration at Branch-2
FlexVPN Hub Configuration at the HQ
Verification on FlexVPN Spoke at Branch-1
Verification on FlexVPN Spoke at Branch-2
Verification on the FlexVPN Hub at HQ
FlexVPN Client NAT to the Server-Assigned IP Address
Configuration on the FlexVPN Client
Verification on the FlexVPN Client
FlexVPN WAN Resiliency, Using Dynamic Tunnel Source
FlexVPN Client Configuration on the Dual-Homed Branch Router
Verification on the FlexVPN Client
FlexVPN Hub Resiliency, Using Backup Peers
FlexVPN Client Configuration on the Branch Router
Verification on the FlexVPN Client
FlexVPN Backup Tunnel, Using Track-Based Tunnel Activation
Verification on the FlexVPN Client
Summary

Part VI IPsec VPN Maintenance

Chapter 13 Monitoring IPsec VPNs

Introduction to Monitoring
Authentication, Authorization, and Accounting (AAA)
NetFlow
Simple Network Management Protocol
VRF-Aware SNMP
Syslog
Monitoring Methodology
IP Connectivity
VPN Tunnel Establishment
Cisco IPsec Flow Monitor MIB
SNMP with IKEv2
Syslog
Pre-Shared Key Authentication

PKI Authentication
EAP Authentication
Authorization Using RADIUS-Based AAA
Data Encryption: SNMP with IPsec
Overlay Routing
Data Usage
Summary
References

Chapter 14 Troubleshooting IPsec VPNs
Introduction
Tools of Troubleshooting
Show Commands
Syslog Messages
Event-Trace Monitoring
Debugging
IKEv2 Debugging
IPsec Debugging
Key Management Interface Debugging
PKI Debugging
Conditional Debugging
IP Connectivity
VPN Tunnel Establishment
IKEv2 Diagnose Error
Troubleshooting the IKE_SA_INIT Exchange
Troubleshooting the IKE_AUTH Exchange
Authentication
Troubleshooting RSA or ECDSA Authentication
Certificate Attributes
Debugging Authentication Using PKI
Certificate Expiry
Matching Peer Using Certificate Maps
Certificate Revocation
Trustpoint Configuration
Trustpoint Selection
Pre-Shared Key
Extensible Authentication Protocol (EAP)

Authorization
Data Encryption
Debugging IPsec
IPsec Anti-Replay
Data Encapsulation
Mismatching GRE Tunnel Keys
Overlay Routing
Static Routing
IKEv2 Routing
Dynamic Routing Protocols
Summary
References

Part VII IPsec Overhead

Chapter 15 IPsec Overhead and Fragmentation

Introduction
Computing the IPsec Overhead
General Considerations
IPsec Mode Overhead (without GRE)
GRE Overhead
Encapsulating Security Payload Overhead
Authentication Header Overhead
Encryption Overhead
Integrity Overhead
Combined-mode Algorithm Overhead
Plaintext MTU
Maximum Overhead
Maximum Encapsulation Security Payload Overhead
Maximum Authentication Header Overhead
Extra Overhead
IPsec and Fragmentation
Maximum Transmission Unit
Fragmentation in IPv4
Fragmentation in IPv6
Path MTU Discovery
TCP MSS Clamping
MSS Refresher

MSS Adjustment
IPsec Fragmentation and PMTUD
Fragmentation on Tunnels
IPsec Only (VTI)
GRE Only
GRE over IPsec
Tunnel PMTUD
The Impact of Fragmentation
Summary
References

Part VIII Migration to IKEv2

Chapter 16 Migration Strategies
Introduction to Migrating to IKEv2 and FlexVPN
Consideration when Migrating to IKEv2
Hardware Limitations
Current VPN Technology
Routing Protocol Selection
Restrictions When Running IKEv1 and IKEv2 Simultaneously
Current Capacity
IP Addresses
Software
Amending the VPN Gateway
Global IKE and IPsec Commands
FlexVPN Features
Familiarization
Client Awareness
Public Key Infrastructure
Internet Protocol Version 6
Authentication
High Availability
Asymmetric Routing
Migration Strategies
Hard Migration
Soft Migration
Soft Migration Example
Migration Verification

Consideration for Topologies
Site-to-Site
Hub and Spoke
Remote Access
Summary

Index

Foreword

Dear reader,

My name is Frederic Detienne and I had the chance to participate during the early days and eventually to lead cryptographic product development in Cisco, acting as Architect of DMVPN and FlexVPN. I started as a TAC Engineer and have evolved into network designer and consultant, inside Cisco and toward our customers. I work across functions with our Engineering, Advanced Services, Support, and Marketing departments.

It seems like yesterday or many eons ago ... In the beginning were crypto maps.

A few dinosaurs (like myself) started their journey in cryptographic protocols and algorithms when Cisco released CET (Cisco Encryption Technology) on IOS 11.2 in August 2003.

My only exposure to cryptography had been strictly theoretical, as a student at the University of Liège 7 years before. I suppose I was lucky to have had such a background as around me, nobody seemed to have received any crypto education nor felt inclined toward that very obscure technology. Before that, cryptography was managed through very complex systems, mostly reserved to governments and militaries.

CET was commercial grade in the sense that it was a major simplification over the former systems. It allowed a mere mortal to configure a very regular and relatively cheap router (Cisco 2500) to encrypt data across a public Layer 3 network. The cryptographic algorithms were very good: DES, then 3-DES, Diffie-Hellman key exchange. At 160 Kbps, the throughput was acceptable in those days.

In the aggregation services, technology goes through 4 steps: make it work, make it work reliably, make it work at speed, make it work at scale. There are other timelines of interest, but this one mattered particularly for cryptographic VPNs.

CET evolved into IKE/ISAKMP + IPsec as the drafts matured into standards under the leadership of Dan Harkins.

Nobody really knew what we were going toward. The initial code inherited from CET which we still had to support for our early adopters. It was also modular and ready to accommodate future enhancements, optimizations and hardware architectures. In a word: it was messy.

The data-plane vs control-plane separation outlined into RFC2408 was both a blessing and a curse. On one hand it brought complexity, on the other, it brought good OSI and code separation without which we may not have survived.

At the control plane level, IKE itself and its rekey complexity, the differences in behavior between IKE SA rekeys and IPsec SA rekeys triggered many race conditions. Overall, we managed to stabilize the system and we “made it to work reliably.” Step 2 was complete.

In the data plane things were less rosy. Crypto maps quickly showed their limits:

–the combinatory explosion of source/destination pairs on ever larger and complex networks

–code complexity due to packets being stolen in OSI layer 2 and re-encapsulated into a new IP header (OSI layer 3)

The security policy size explosion made mesh networks totally unmanageable. Besides, the security policy was mostly a static transcription of information we already had in the dynamic routing table, which led to customer frustration.

A few site-to-site and hub-and-spoke configurations were possible, but manageability and scalability suffered badly. The TAC was recommending the use of GRE protected by IPsec in order to run routing protocols on top of the tunnels. This quickly became the preferred way to deploy complex meshes. Scalability was relatively limited due to hardware performance, but it really made everyone's life easier and the support of those network became very pleasant with more and more satisfied customers.

Meanwhile, EasyVPN had appeared and was offering a remote access solution. The clients were either PC software or small branch routers. The big advantage was that the hub configuration was very compact—a few lines would allow hundreds of remote branches to connect. Unfortunately, the underlying implementation relied on crypto maps and suffered from quality and supportability issues. While EasyVPN was very good, it was not stable enough compared to the GRE/IPsec solution we used in mesh.

Customers had to choose between easy of configuration for large but simple hub-and-spoke networks and a more complex configuration for mesh networks.

One day, someone showed me NHRP: a protocol to establish circuits on demand. The code was very crude and incomplete but the developer (who had left Cisco by then) had provisioned for GRE tunnels, very likely in order to test his code without expensive equipment. I had this light bulb moment and hacked together a prototype to encrypt those GRE tunnels as they were created.

DMVPN was born in a TAC lab in Brussels, demonstrated to our colleagues in San Jose, California, and developed into a product.

We now had something that worked well, was satisfactorily stable despite being a fresh feature, and offered an easy configuration for complex networks. It started as DMVPN phase 1 with hub-and-spoke only and followed quickly with DMVPN phase 2 allowing dynamic branch-to-branch tunnel creation.

Scale was not there yet though. The IGPs (OSPF and EIGRP mostly) caused significant burdens and deploying more than 350 nodes networks was still a burden. It may seem small today, but the bulk of the network sizes grew as technology permitted. A mesh network of 350 nodes was fantastic back then. Just that the market quickly got used to it and demand for more appeared.

The market demanded that we scale up both the tunnel density (the number of tunnels on a single given platform) and horizontally (the ability of a cluster of DMVPN hubs to collaborate to service). DMVPN phase 2 daisy chaining was a dreaded system to design and troubleshoot. Besides scalability problems, it also suffered from reconvergence time and convoluted configuration.

The workload reduced dramatically while market shares and revenue took off; we started work on scaling DMVPN before it become too stringent.

The semantic of the NHRP redirect and NHRP resolution forwarding appeared and helped us scale almost limitlessly across hubs. You could literally have dozens of hubs working in cluster mode. Also importantly, we could finally get away from the traditional IGPs and investigate lighter protocols such as RIP, OTV and even BGP (which is feature-rich and complex at large but out of which we only needed the simplest elements). We could now scale to about 1500 peers per hub and a virtually limitless number of hubs. Each additional hub would linearly add its capability to the cluster. This was an important step forward in network design.

The biggest challenge was now to educate our customers and sales team about the various options and design. When you work on Crypto VPNs all day, every day, it is easy to forget that very few people

actually understand the ins and outs of every feature and design. Also, many customers were satisfied with what they had and had no reason to investigate for more—or even suspect that something better could exist.

A metric of complexity could be seen in our 8 hours CiscoLive session going over our multiple Crypto VPN solutions and their use case:

- Crypto maps
- Easy VPN (client mode and network extension mode)
- Enhanced Easy VPN
- GRE/IPsec
- GET VPN
- DMVPN phase 1, 2 and 3

The pros and cons panned out as below:

- Crypto maps were still as limited and terrible as before but are necessary for third-party integration as they offer compatibility with devices that have minimal functionality.
- EasyVPN supported remote access (especially the software client) compact but the feature had grown organically and the UI was terrible; it was also crypto map based, and its quality was poor.
- Enhanced Easy VPN solved the crypto map problem and was a major improvement over Easy VPN, but it did not enjoy proper marketing and remained poorly adopted. The UI was the same and hence difficult.
- GRE/IPsec was slowly disappearing at the benefit of DMVPN and tunnel protection in the site-to-site scenarios.
- GET VPN has lower security and limited scalability, but it is lighter on resources when used properly, if the use case is adequate. Notably, it allows native multicast.
- DMVPN was growing in both the hub-and-spoke and partial mesh cases, but the routing protocol was a deterrent for Security Operations who preferred using EasyVPN.

This really meant 8 hours during which we barely had the time to describe how a solution worked and what use case it was best for.

Customers who were successively shopping for a remote access solution, then a site-to-site, then a dynamic mesh ... had to study and learn new ways of designing and troubleshooting for each feature, over and again.

The complexity we were witnessing in TAC on our fresh recruits was impacting our customers, partners, Advanced Services, and sales teams.

At the same time, as all things so far, after a few years, market demand slowly started to outgrow DMVPN. Tunnel density still had to increase and the routing protocols were not scaling anymore.

We decided to merge EasyVPN and DMVPN features into a single feature that would offer us the advantages of both under a single feature set: one time learning, applicable always. The characteristics had to be the following:

- clear, consistent, compact, and powerful CLI: simple things ought to be simple to configure, complex things ought to be possible.

- using routing protocols should be a customer choice, not mandatory.
- NHRP usage could decrease except for spoke-spoke tunnel creation
- increased scale to 10,000 tunnels per hub at least.
- all the remote access management features had to be applicable to site-to-site and hub-and-spoke (AAA authorization in particular to apply per user QoS, ACLs, and so on.)
- reduce the reliance on PKI and make pre-shared keys more manageable. Both had to be possible, at least for hub-and-spoke
- backup and load balancing scenario
- third-party interoperability
- high serviceability/troubleshootability
- reduced learning time by using consistent protocol and data flows
- state of the art security at the cryptographic and network level

Because we could not take the risk to break IKEv1 stability nor invest in a protocol that was slated to disappear, we used IKEv2 as an inflection point to do things right. Clean implementation, clean user interface.

Today, we are capable of offering combined training, including hands-on experience, covering remote-access, hub-and-spoke, dynamic mesh, AAA management, and some troubleshooting in 4 (fours) hours. The total training time has decreased by an order of magnitude.

FlexVPN is not perfect and is not the end of the road, but in terms of applicability and total cost of ownership, taking in account training time and supportability, this is the best we have ever had.

I hope you will have as much pleasure discovering FlexVPN in this book as we had developing those features, thinking about you, our users, our customers, our sponsors.

None of this would have happened without great individuals who went beyond the basic market analysis that a typical Product Management team performs and took it on themselves to listen to our customers' real demands.

Namely, it took the courage of one Senior Manager, Pratima Sethi, to sponsor and execute on the development of FlexVPN. She also made DMVPN and EasyVPN successful; she understood deeply the need of post-deployment capabilities such as monitoring and troubleshooting and made it all possible.

The authors of this book, Amjad Inamdar and Graham Bartlett, are long-time collaborators who also deeply impacted all our VPN solutions, and I am very proud to work with them.

The teams prominent members included Alexandre Honore, Olivier Pelerin, Wen Zhang, Raffaele Brancaleoni, Sairam Yeleshwarapu, Saikrishna Adoni, Tapesh Maheshwari, Raghunandan P., and many others to whom I apologize for not citing.

Frederic Detienne

Distinguished Services Engineer

Cisco

Icons Used in This Book



Router



Workgroup Switch



Private Key



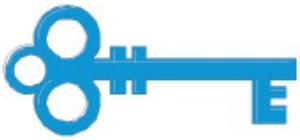
Internet/Cloud



Home Office



Office



Key





Server

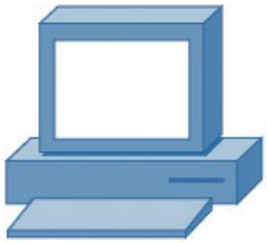


Printer





Laptop



PC



Phone

Command Syntax Conventions

The conventions used to present command syntax in this book are the same conventions used in the IOS Command Reference. The Command Reference describes these conventions as follows:

- **Boldface** indicates commands and keywords that are entered literally as shown. In actual configuration examples and output (not general command syntax), boldface indicates commands that are manually input by the user (such as a **show** command).
- *Italics* indicate arguments for which you supply actual values.
- Vertical bars (|) separate alternative, mutually exclusive elements.
- Square brackets [ ] indicate optional elements.
- Braces { } indicate a required choice.
- Braces within brackets [{ }] indicate a required choice within an optional element.

Introduction

The motivation for writing this book was to educate users and customers about the benefits that FlexVPN and IKEv2 bring and provide an in-depth coverage of the building blocks and topics related to IPsec VPNs in general, in an easy-to-understand manner. FlexVPN was a breath of fresh air with regards to VPN technologies; for the first time all VPN technologies could be configured under a single CLI construct. We want to educate users so that secure, efficient VPN technologies can be implemented not only using Cisco IOS, but with third-party equipment also.

FlexVPN has allowed IKEv2 and IPsec VPNs on Cisco IOS to become a lot more user friendly; IKE, IPsec, concepts of cryptography, and VPNs can be hard subjects to understand. This book is intended to explain these topics and allow the reader to not only grasp the concepts, but master them.

The books explains how IPsec VPNs deal with NAT traversal, fragmentation, segmentation, IP dual stack, multicast, non-IP protocols and so on.

When VPNs are configured, there are a plethora of options; this book is intended to clarify these with ample illustrations and configuration examples so technologies are implemented in a secure and streamlined fashion.

When we talk to customers, many are unaware of what's happening under the hood and what impact a certain command will have. This book tries to clarify these points.

Goals and Methods

Provide a guide that will take the reader from knowing very little about VPN technologies to having an in-depth understanding.

Prevent customers from making mistakes that lead to network down scenarios or put the overall architecture at risk.

Give architects an understanding of the technology, allowing them to design VPN systems that meet business needs.

Give designers the knowledge where to position certain features.

Give implementors the understanding of how various features work along with end-to-end configurations examples that can be used as reference.

Give support staff an understanding of the protocols, configuring them, and how features integrate. This will result in a deep understanding, enabling timely debugging and troubleshooting.

Provide Security Operation Center guidance on telemetry that can be gained when an IKE and IPsec SA are created. This provides a methodology to perform monitoring and troubleshooting.

Provide advice and guidance on how to migrate existing IKEv1 architecture to using IKEv2.

Allow accreditors to understanding technologies, resulting in assurance that the architecture presented will meet the intended security requirements.

Give project managers an understanding of the components required to perform migrations from IKEv1 to IKEv2.

Who Should Read This Book?

Anyone that is involved with the lifecycle of deploying an IPsec VPN. This includes architects, designers, security engineers, support engineers, accreditors, and members of a SOC/NOC.

This book tries to explain the protocols at an RFC level, so it will provide the reader with an understanding that is not just specific to Cisco, but is applicable to any standards-based implementation.

For any individual that is developing services that are consumed by an IPsec VPN architecture (RADIUS, PKI, and similar ones), this book allows the reader to understand the protocol flows and the interaction between IKEv2/IPsec and their services.

VPN technologies are an integral part of the many of the Cisco certification tracks. This book would serve as a valuable study aid providing an in-depth coverage of the IPsec VPN foundational topics in an easy to understand manner. Some of such certification tracks are

- Security—CCNA, CCNP, and CCIE
- Routing and Switching—CCNA, CCNP, and CCIE
- Design—CCDA, CCDP, and CCDE
- Service provider—CCIE

Simply put—if you want to understand IPsec VPN building blocks and architectures, and deploying IPsec VPNs when using IKEv2 this book is for you.

How This Book is Organized

Chapter 1 Introduction to IPsec VPNs

This chapter describes the purpose of VPNs and the types of cryptography (symmetric and asymmetric). We cover cryptographic protocols used in the generation of IPsec VPNs.

We explain how confidentiality and integrity are achieved using Encapsulation Security Payload (ESP) and how integrity is achieved using Authentication Header (AH).

We introduce IKE and IPsec and the relationship that these have.

This chapter describes the components that make up IPsec, including the Security Parameter Index (SPI), Security Policy Database (SPD), Security Association Database (SADB), Peer Authorization Database (PAD), lifetime, and sequence numbers. We explain how these are interlinked and what relationship exists.

The two modes of IPsec, tunnel and transport, are described. We explain the benefits of each. The benefits of ESP version 3 are described.

Chapter 2 IKEv2: The Protocol

The Internet Key Exchange (IKE) protocol is described in detail. The format of the IKE header and the various packet exchanges (IKE_SA_INIT, IKE_AUTH, INFORMATIONAL, CREATE_CHILD_SA) are described. You will understand how the IKE SA is created and the components that are used to construct this, such as key material generation.

Features of IKEv2, such as anti-replay, the anti-DDoS cookie, configuration payload, and acknowledged responses, are described along with the protocols used by IKE; encryption, integrity,

PRF, and Diffie-Hellman are listed.

This chapter details how IKEv2 operates when NAT is used on the transport network. The various keepalive mechanisms are covered, including IKE and NAT keepalives. This chapter covers a number of additional IKEv2 related RFCs.

Chapter 3 Comparison of IKEv1 and IKEv2

Within this chapter the history of IPsec and IKEv1 is covered, including all the RFCs (2401 to 2412) that were created to define the implementation of IKEv1-based IPsec VPNs. The key similarities and the key differences of IKEv2 compared to IKEv1 are covered, including exchange modes, authentication, use of identities, anti-DDoS, lifetimes, and many more topics.

Chapter 4 IOS IPsec Implementation

The specific types of VPN implementation of Cisco IOS and IOS-XE are introduced. This chapter describes how to implement tunnel or transport mode. The two encapsulation types, GRE and VTI, are described, along with their benefits and limitations. The various implementation modes (dual stack, mixed mode, and auto) are covered. We also introduce VRF-aware IPsec.

Chapter 5 IKEv2 Configuration

This chapter contains an overview of the IKEv2 configuration features and how these interoperate. The various components of IKEv2 are covered: IKEv2 proposal, IKEv2 policy, IKEv2 profile, IKEv2 keyring, and the IKEv2 global configuration. We also cover other components that are critical to configuring IPsec VPNs, such as PKI and IPsec. The powerful pre-configured attributes are introduced, and their benefits are explained.

Chapter 6 Advanced IKEv2 Features

This chapter covers IKEv2 advanced features, including some that are not part of the standard IKEv2 RFC. IKEv2 fragmentation and the transportation of Security Group Tags (SGT) are described, along with the methods to delete a session should a peer be revoked or the peer's certificate expire. The lifetime of the IKEv2 session is examined and the effect this has is described in detail.

Chapter 7 IKEv2 Deployments

This chapter describes a number of scenarios to give the reader an understanding of the various types of IKEv2 deployments. Both IPv4 and IPv6 are covered, with authentication using pre-shared keys, RSA certificates, ECDSA certificates, and HTTP URL Cert. The IKE anti-DDoS mechanism is illustrated in detail.

Chapter 8 Introduction to FlexVPN

After an overview of FlexVPN, the tunnel interface types (static, virtual-template, and virtual-access) and IOS AAA infrastructure are described in detail. The building blocks of FlexVPN—Name Mangler, IKEv2 authorization policy, with user, group. and implicit authorization—are described. The configuration exchange is illustrated, along with advertising prefixes using IKEv2 routing.

Chapter 9 FlexVPN Server

The chapter provides an overview of FlexVPN Server. EAP authentication is described in detail, along with AAA-based pre-shared keys. Deriving virtual-access interfaces from virtual-templates is illustrated, along with automatic detection of the tunnel mode and encapsulation type using mode auto. RADIUS Packet of Disconnect and Change of Authorization (CoA) are described. The IKEv2 auto-reconnect, AnyConnect-EAP, and dual-factor authentication features are described. The FlexVPN Server supported clients are covered.

Chapter 10 FlexVPN Client

This chapter begins with an overview of FlexVPN Client. EAP authentication is described in detail. Client-specific attributes are described: split-DNS, WINS, and Domain Name. The FlexVPN client profile is described. The following specific features of FlexVPN client are illustrated: Backup gateways, dial backup, backup groups, tunnel interface types, tunnel initiation types, and FlexVPN with NAT. This chapter describes design considerations and troubleshooting specific to the FlexVPN client.

Chapter 11 FlexVPN Load Balancer

This chapter presents an overview of the FlexVPN Load Balancer. It details the core components and RFC5685 “IKEv2 Redirect and Hot Standby Routing Protocol.” How the cluster operates, including cluster load, is detailed. FlexVPN client and server configurations are illustrated. Troubleshooting a specific FlexVPN load balancer configuration is described, and a number of example configurations are shown.

Chapter 12 FlexVPN Deployments

This chapter contains a number of example scenarios which illustrate the following FlexVPN deployments: AAA-based pre-shared key, user and group authorization, FlexVPN routing with dual-stack and tunnel mode auto, NAT with server-assigned IP addresses, WAN resiliency using dynamic tunnel source, hub resiliency using backup peers, and FlexVPN backup tunnel using track-based activation.

Chapter 13 Monitoring IPsec VPNs

This chapter describes common methods for monitoring IPsec VPNs using AAA, SNMP, and syslog. A monitoring methodology is described that covers IP connectivity, VPN tunnel establishment, authentication, authorization, data encapsulation, data encryption, and overlay routing.

Chapter 14 Troubleshooting IPsec VPNs

This chapter describes the tools of troubleshooting: Event Trace Monitoring, IKEv2, IPsec, KMI, and conditional debugging. Troubleshooting steps are described for IP connectivity, VPN tunnel establishment, authentication, authorization, data encapsulation, data encryption, and overlay routing.

Chapter 15 IPsec Overhead and Fragmentation

This chapter describes computing IPsec overhead for ESP and AH and the effect that IPsec and fragmentation have for both IPv4 and IPv6. The following topics are illustrated: Path MTU Discovery (PMTUD), TCP MSS clamping, fragmentation, and PMTUD (specifically on tunnel interfaces). The impact of fragmentation is described.

Chapter 16 Migration Strategies

The chapter illustrates the considerations when migrating from IKEv1 to IKEv2. It covers hardware, VPN technologies, routing protocols, restrictions for IKEv1 and IKEv2, capacity planning, global commands, FlexVPN features, PKI authentication, high availability, and asymmetric routing. Migration strategies for hard and soft migrations are covered. It also discusses considerations for specific topologies: site-to-site, hub-and-spoke, and remote access. There is also an example migration scenario.

Part I: Understanding IPsec VPNs

Chapter 1. Introduction to IPsec VPNs

This chapter introduces the concepts of Virtual Private Networks (VPNs), focusing specifically on Internet Protocol Security (IPsec); this chapter will also introduce the Internet Key Exchange protocol version 2 (IKEv2).

Ten years ago IPsec VPNs were commonly employed only for connectivity between site locations or for remote mobile workers requiring secure connectivity from hotels, where specialist software was required. With the advance of the Internet of Things, the number of devices that require secure connectivity over insecure networks is almost endless; the ability to connect, communicate with, and remotely manage every IP-enabled device over a secure medium has increased the demand for VPN technologies. No longer can devices be configured to use insecure protocols across the Internet; the mandatory method of communication requires security, and for this reason the use of IPsec VPNs has increased in recent years, with demand expected to continue rising.

This chapter describes the components that are used to create IPsec VPNs and will make the connection between Security Protocols and the IPsec Security Associations. Security services provided by IPsec are described, along with how the components of IPsec operate, with specific focus on the Security Association Database, Security Policy Database, and Peer Authorization Database.

The cryptographic components of IKEv2 are introduced to provide an understanding of what is happening internally when creating a secure authenticated IPsec Security Association. Asymmetric cryptography is described, as well as the way it is used within the establishment of an IPsec Security Association. Authentication using digital signatures, and shared secrets is covered, along with the construction of shared secret key material. Symmetric cryptography is described and how this relates to both Authentication Header and Encapsulation Security Payload and the services that these provide.

Both modes of IPsec are discussed, tunnel mode and transport mode, and the benefits that these bring.

The Need and Purpose of IPsec VPNs

A VPN allows users to securely extend a private network across an untrusted network, such as the Internet. The term “Virtual” emphasizes that the private network is virtually extended by a logical private connection. VPN peers are not mandated to be physically connected. A VPN allows a network-enabled device to securely send and receive data across unsecured networks, as if it were directly connected to the private network.

When IPsec VPNs are used, traffic will be protected to ensure that an observer cannot view the plaintext data; this is achieved by encryption that provides confidentiality. An intruder cannot tamper with the data. This is achieved by cryptographic hashing and signing of the data exchanged, which provides integrity. A VPN must be established only with a chosen peer; this is achieved using mutual authentication.

Please be aware of the differences between a VPN using IPsec and a VPN using Multiprotocol Label Switching (MPLS). MPLS uses labels to differentiate traffic sent on the wire; this is a method to separate traffic, but unlike IPsec, it offers no confidentiality or integrity protection.

At the time of writing, almost every operating system ships with an IPsec VPN client, and there are

countless hardware devices that provide various forms of IPsec VPN gateway functionality. IPsec VPNs are a popular choice now for secure connectivity over the Internet or for delivering secure communications over untrusted networks, such as the Internet.

Internet Key Exchange (IKE) creates one or more pairs of IPsec Security Associations (SA). The SAs are what data will be sent within; data is encapsulated and encrypted prior to being sent onto the wire. The cryptographic key material that is used to protect the IPsec Security Association is negotiated in the IKE exchange.

The IPsec Security Association has a defined lifetime. It will expire after a set duration, volume of data, or both.

Building Blocks of IPsec

A number of components are used to create and maintain an IPsec session. These components integrate together to provide the required security services that protect traffic.

IPsec establishes tunnels between endpoints; these can also be described as *peers*. The tunnel can be protected by various means, such as integrity protection and/or confidentiality protection.

The following sections describe at a high level each of the building blocks that are used to construct and maintain IPsec sessions.

Security Protocols

IPsec provides security services through the use of two protocols, the Authentication Header and Encapsulating Security Payload. Both protocols use cryptographic algorithms for authenticated integrity services; Encapsulation Security Payload provides encryption services in combination with authenticated integrity.

Both protocols support two modes of use: transport mode and tunnel mode. When transport mode is used, both Authentication Header and Encapsulation Security Payload provide protection primarily for protocols that are sent by the IPsec peer. Originally this was a host device, but more recently it can be a network device also. When tunnel mode is used, Authentication Header and Encapsulation Security Payload are applied to tunnelled IP packets.

Security Associations

An IPsec Security Association (SA) is the secure session created between two peers. This connection is unidirectional and provides security services to the traffic carried by it. To secure typical bidirectional communication between two IPsec-enabled systems, a pair of SAs (one in each direction) is required.

The SA is protected by one of the security protocols described above.

Each SA is identified by a Security Parameter Index (SPI). This is a unique 32-bit value that is included in the security protocol header.

Key Management Protocol

The creation and maintenance of the IPsec Security Association is controlled by a key management protocol. RFC4301 actually allows for IPsec to be manually keyed; however, for scalability and security purposes a SA management protocol is required. The Internet Key Exchange (IKE) is used to create a maintain IPsec Security Associations.

IPsec Security Services

IPsec provides cryptographically based security for IPv4 and IPv6 traffic. The set of security services offered includes access control, connectionless integrity, data origin authentication, anti-replay services, confidentiality (via encryption), and traffic flow confidentiality.

IPsec security services require the use of cryptographic keys, which are negotiated using a separate function that is achieved by the Internet Key Exchange.

The majority of the security services are provided through use of either of the two traffic security protocols: Authentication Header and Encapsulating Security Payload.

IPsec creates a boundary between unprotected and protected interfaces for a host or a network device. Traffic traversing this boundary is subject to enforcement of the security policy. The security policy indicates whether packets cross the boundary without intervention, are afforded security services via Authentication Header or Encapsulation Security Payload, or are discarded.

It should be noted that because IPsec is an open standard, these services can be achieved in an interoperable manner.

Access Control

Access control is achieved by specifying what traffic should be protected, allowed in the clear, or discarded. Enforcement of this security policy can be applied both in the outgoing direction (what traffic will be protected) and in the incoming (what traffic should be received). Traffic selectors are used to define the traffic that each SA protects. It should be noted that access control is not a firewall function but allows for defined traffic to be discarded.

Anti-replay Services

The method to mitigate an attacker replaying traffic is achieved using sequence numbering of traffic. Once a packet has been successfully verified, the sequence number cannot be received again. A window of sequence numbers is permitted, with only packets falling within this window being processed.

Confidentiality

Confidentiality is achieved by transforming data from an intelligible form (plaintext) into an unintelligible form (ciphertext). This is achieved using Encapsulated Security Payload, which provides confidentiality services when the sending host encrypts data and a receiving host decrypts it, which turns ciphertext back into plaintext.

Connectionless Integrity

Connectionless integrity is a service that detects modification of an individual IP datagram, without regard to the ordering of the datagram in a stream of traffic. This is provided by both Authentication Header and Encapsulated Security Payload services. Hashed Message Authentication Code (HMAC) provides connectionless integrity by way of an Integrity Check Value (ICV). This differs from connection-oriented integrity, which has the ability to detect packet loss and includes the ability to re-sequence received packets.

Data Origin Authentication

Data origin authentication verifies the identity of the claimed source of data. The Internet Key Exchange provides a method of authentication. Both Authentication Header and Encapsulated Security Payload services provide connectionless integrity, which integrates with data origin authentication.

Traffic Flow Confidentiality

Traffic Flow Confidentiality (TFC) is the ability to conceal source and destination addresses, message length, or frequency of packets sent. The use of tunnel mode allows the inner IP headers to be encrypted, concealing the identities of the (inner) traffic source and destination. Encapsulation Security Payload payload padding can be used to hide the size of the protected packets, further concealing the characteristics of the protected traffic. Encapsulation Security Payload version 3 allows for “dummy” packets to be sent over the IPsec Security Association; these are packets that when decrypted contain the next header of protocol 59, which denotes that there is no next header and the packet should be discarded by the receiver.

Components of IPsec

When an IPsec Security Association is created, a number of components are used to process plaintext traffic that is to be protected and subsequently turned into ciphertext. The same components are used when ciphertext traffic that requires decryption is received. Three databases are used by IPsec; although these are separate databases, each shares a relationship with the others. The Security Policy Database (SPD) defines what traffic is to be protected. The Security Association Database (SAD) defines how traffic is to be protected. The Peer Authorization Database (PAD) provides a mechanism to enforce policy based on IKE identity.

Security Parameter Index

The Security Parameters Index (SPI) is a randomly generated 32-bit value that is used by a receiver to identify the SA to which an incoming packet should be bound. The SPI is carried in Authentication Header and Encapsulation Security Payload protocol headers to enable the receiver to uniquely identify an SA. For cases in which multiple SAs exist between two peers, the SPI must be unique. When a device has SAs with multiple different peers, there is a case that the same SPI could be used, for this reason the lookup to the SAD will also use source and destination IP address.

Security Policy Database

The Security Policy Database (SPD) defines the IPsec boundary between protected and unprotected interfaces on an IPsec peer and specifies security policy for traffic crossing the boundary.

The SPD security policy consists of a processing action; this action can be “protect,” “discard,” or “bypass.” Usually management traffic, such as IKE, would have the “bypass” action, and data traffic would have the “protect” action. The security parameters for the “protect” action include the security protocol (Encapsulation Security Payload or Authentication Header), the protocol mode (transport or tunnel), cryptographic algorithms, local and remote IPsec peers, and the granularity of the SA, based on whether the SA traffic selectors are derived from the parent SPD entry or from the packet that matched the SPD (when each packet flow creates an IPsec Security Association).

The SPD is an ordered first-match database of SPD entries in which each SPD entry defines the security policy for a traffic flow. An IPsec peer can have one or more instances of an SPD.

> **Note**
> The Cisco IOS IPsec implementation has an SPD instance per IPsec enabled interface.

For plaintext outbound traffic, after processing by the SAD, the SPD is searched to determine if the packet should be protected.

For inbound traffic (protected traffic received), after the SA is determined via the SAD, the SPD is searched to check if the decrypted/authenticated packet was supposed to be protected.

Security Association Database

The Security Association Database (SAD) contains parameters associated with each unidirectional IPsec Security Association. Each SA entry consists of the negotiated traffic flow protected by the SA and negotiated security parameters, including the security protocol (Encapsulation Security Payload or Authentication Header), protocol mode (tunnel/transport), cryptographic algorithms and local/remote IPsec peer addresses. The SA entry also defines SA lifetime, sequence number counter, anti-replay window, and Security Parameter Index (SPI), which for an outbound SA is included in the Encapsulation Security Payload or Authentication Header and for the inbound SA is used to look up the SA using the SPI in the received Encapsulation Security Payload or Authentication Header.

An IPsec peer has an SAD instance per SPD instance.

> **Note**
> The Cisco IOS IPsec implementation has an SPD and SAD instance per IPsec enabled interface.

For outbound traffic, the SAD is searched to see how the packet should be protected.

Peer Authorization Database

The Peer Authorization Database (PAD) links the Security Policy Database with the IPsec Security Association management protocol such as IKE. The PAD defines the authorized list of IPsec peers, identified by their IKE identities, that can establish SAs with a local SPD instance. Each PAD entry defines an authorized peer or a group of peers and the authentication parameters for those peers. The authentication parameters include the authentication protocol (IKEv1, IKEv2, etc.), authentication method (shared-secret, certificate-based, etc.), authentication credentials, and other parameters such as certificate revocation link.

> **Note**
>
> In Cisco's IOS IPsec implementation, the IKE profile embodies the functionality of the PAD. An IKE profile has a list of allowed peer IKE identities, authentication method, and link to authentication credentials. Attaching the IKE profile to an IPsec-enabled interface (SPD instance) defines the PAD policy for that interface.

Lifetime

The lifetime of the Security Association can be set by a time-based method, the amount of traffic that is sent by the SA, or both. A lifetime is defined because the requirement that certain encryption ciphers are only secure when protecting a finite amount of traffic. Before this time ends, the SA must be replaced with a new SA (and new SPI) or terminated. The new SA ensures that security is maintained. If the system uses a time-based method and volume-based lifetime, whichever condition is met first will ensure that the SA is rebuilt. We sometimes use the term *rekey* for this behavior.

Cryptography Used in IPsec VPNs

The purpose of a Virtual Private Network is to offer secure connectivity between entities. It is commonplace for this connectivity to provide confidentiality and integrity.

Within the IKE exchange, both peers negotiate cryptographic algorithms that are later used to protect the session and exchange Diffie-Hellman public key values; from this exchange a shared secret is derived, which is then used as the input to a function to generate key material used to protect subsequent IKE exchanges.

Four cryptographic algorithms are negotiated: an encryption algorithm, an integrity algorithm, a Diffie-Hellman public value/group, and a pseudorandom function (PRF).

The PRF is a method to generate a value that appears random from a given input. It is used for the construction of key material used for all of the cryptographic algorithms in both the IKE Security Associations and IPsec Security Associations and also some methods of authentication.

After the session has been secured, both parties must authenticate themselves. This can be achieved using either EAP (IKEv2 only), shared-secrets, or digital signatures. The methods for using EAP and shared secrets are covered in chapter 2, "IKEv2: The Protocol."

IKEv2 uses both asymmetric and symmetric cryptography.

Symmetric Cryptography

An encryption algorithm (which is also referred to as a cipher) is a method of transforming plaintext into ciphertext under the control of a secret key.

This process of transformation is called encryption or encipherment. The reverse process, during which ciphertext is transformed back into plaintext, is called decryption or decipherment.

The process requires that each party has access to the secret key. This secret key needs to be known to both sides and is required to be kept secret. Encryption algorithms that have this property are called symmetric cryptosystems or secret key cryptosystems.

Asymmetric Cryptography

Asymmetric cryptography (also known as public-key cryptography) is a method of cryptographic protocols based on algorithms that require two separate keys, one of which is secret (or private) and one of which is public and shared. The two keys in the key pair are mathematically linked, and the strength of the algorithm is based on the fact that no efficient solutions are known to derive the private key from the public key: examples of these include factorization, discrete logarithm, and elliptic curve functions.

Each key will perform the inverse of the other. For example, a private key is used to encrypt plaintext or to create a digital signature, whereas the associated public key is used in the opposite operation, in these examples to decrypt ciphertext or to verify a digital signature. When digital signatures are used, one peer will sign a known value with its private key, and the opposing peer will validate this with the cryptographically linked public key, which is included in the certificate of the peer that is signing with the private key.

When an IKE or IPsec Security Association is created, the key used as the input to the encryption function is known as SK_e (the letter “e” indicating encryption). The same key is used as input to the decryption function by the opposing peer, which aligns with the logic behind symmetric encryption, where a single key is used for both encryption and decryption. The bulk of data is encrypted using a symmetric algorithm because asymmetric encryption is computationally expensive compared to symmetric. Symmetric algorithms can be easily accelerated by hardware, unlike asymmetric cryptographic algorithms, which can be accelerated but commonly are not.

The Diffie-Hellman exchange is a form of asymmetric cryptography; the details are covered within the next section. It is important to note that data is not encrypted or signed by a public or private key, but rather a shared secret is derived between two parties using large prime numbers. However, this is still considered to be public key cryptography.

The Diffie-Hellman Exchange

When a secure authenticated connection is established between two parties, as IKE needs, a method is required to secure the channel and then allow both parties to prove their authenticity. Establishing a shared secret to protect the channel is achieved by the Diffie-Hellman exchange. The Diffie-Hellman exchange is used by two parties to generate a shared secret over an insecure medium. It is a mathematical function that is easy to do one way, but it is very hard to reverse; with sufficiently large values, it is computationally almost impossible to reverse it without having access to a piece of information (which is kept secret).

The Diffie-Hellman exchange is susceptible to a man-in-the-middle attack, where malicious

observers can substitute their own values and create two separate secure channels with each party. For this reason, when IPsec VPNs are used, an additional authentication phase occurs to ensure that a man-in-the-middle attack is not possible.

The idea of the Diffie-Hellman exchange was first presented to the public in a paper by Whitfield Diffie and Martin Hellman entitled "New Directions in Cryptography" in 1976. However mathematicians in the United Kingdom's GCHQ had developed the same concept earlier, although this research at the time was classified and not publicly released.

The following illustrates a typical Diffie-Hellman exchange, the public value being denoted by 'A' and 'B'; these values are very small and are used as an example.

$p = 23$

$g = 5$

p and g are constants defined by the Diffie-Hellman group in use; the value can be seen in RFCs that describe the Diffie-Hellman parameters. The value p is a prime and g is the generator. The reason that these values are defined and not generated by each peer is that they are used as a constant in the value of the shared secret. Each peer selects only a local public and private value. As the value p is used as a modulus, in this example it gives 23 possible values. Standard implementations use the value of p in 100s of digits, which makes them computationally infeasible to reverse engineer. This is known as the discreet logarithm problem. The value of 2 for the generator is used because the computation is efficient on a binary-based machine and it results in a uniform distribution.

The peers, otherwise known as Alice and Bob, calculate a private value.

Alice calculates her private value to be "a"; Alice then calculates her public value "A" using the formula

$$A = g^a \bmod p$$

In this instance Alice calculates a = 6,

$$A = 5^6 \bmod 23$$

$$A = 15625 \bmod 23$$

$$A = 8$$

Alice sends 8 as her public value to her peer, Bob.

Note that for a malicious observer to derive Alice's secret value, they would have to perform all calculations until they too landed with the public value of 8, which would reveal the secret value of 6. When large numbers are used, this is computationally impossible, even with cloud-based computing.

Bob calculates his private value "b" as 15 and then calculates his public value "B" as

$$B = g^b \bmod p$$

$$B = 5^{15} \bmod 23$$

$$B = 30{,}517{,}578{,}125 \bmod 23$$

$$B = 19$$

Bob then sends his public value 19 to Alice.

Table 1-1 illustrates the generation of the shared secret by both Alice and Bob. Both parties derive the same shared secret.

Alice	Bob
$s = B^a \bmod p$	$s = A^b \bmod p$
$s = 19^6 \bmod 23$	$s = 8^{15} \bmod 23$
$s = 47{,}045{,}881 \bmod 23$	$s = 35{,}184{,}372{,}088{,}832 \bmod 23$
$s = 2$	$s = 2$

Table 1-1 *Alice and Bob Derive Shared Secret*

It can be seen that $(g^a \bmod p)^b \bmod p$ equals $(g^b \bmod p)^a \bmod p$ equals $=g^{ab} \bmod p$.

Both Alice and Bob have arrived at the same shared secret; however, anyone monitoring the network exchange will have seen only the public values that were transmitted, so as long as the private values stay secret, the exchange will be computationally secure. The phrase "computationally secure" is used to indicate where the best possible algorithm for obtaining the private value from the public value requires X operations, where X is such a large number that it is computationally infeasible to perform this many operations by an attacker.

Diffie-Hellman can also be used with an elliptic curve, known as ECDH (Elliptic Curve Diffie Hellman). In an ECDH exchange, points on an elliptic curve are sent between parties in a manner similar to the example above. If ECDH is used, a known elliptic curve would be defined, such as;

$$Y^2 = X^3 + X - 3$$

A base point (G) on the curve will be agreed to before the exchange, for example G = (1,76) denoting the X and Y coordinates on the elliptic curve. Each peer calculates a private value (the peers are referred to as Alice and Bob).

Alice selects a private value *a*, which is used with the base point G. In this case a = 23, and the result is a coordinate on the elliptic curve, A = (2,150), which is sent to Bob.

Bob selects a private value *b*, and using the base point G, he obtains the coordinates of his public value, B (123,187), which is sent to Alice.

Alice can then use her private value (23) and Bob's public coordinates (2,150) to obtain a shared point on the elliptic curve. Likewise, Bob can use his private value (86) and Alice's public coordinates (2,150) to obtain a shared point on the elliptic curve.

Alice's computation = [a]B = [23](123,187) = (156, 75)

Bob's computation = [b]A = [86](2,150) = (156, 75)

The shared secret is then taken to be the x-coordinate (in this case the value 156) of the computed point.

When ECDH is used instead of modulo Diffie-Hellman, the size of the exchange attribute will be considerably smaller for ECDH. Not only are the sizes smaller, the cryptographic workrate is considerably less too. You get more effective use of your resources with regards to the size of key compared to cryptographic security, so for this reason (at the time of this writing) Cisco recommends, whenever it is possible, the use of Next Generation Encryption, which would use Diffie-Hellman Groups 19, 20, or 21.

The hardness of the Diffie-Hellman exchange is based on discrete logarithm problem, whereby obtaining the private key used from only knowing the public value is defined.

The following example shows an IKEv2 key exchange payload from an IKE_SA_INIT exchange; this uses the smallest Diffie-Hellman group (Group 1). Although the key exchange data (public key) looks large, it is in cryptographic terms rather weak, and the use of anything below Diffie-Hellman Group 5 is discouraged, because the small key sizes are not sufficient in light of today's computational processing power.

Click here to view code image

```
      Type Payload: Key Exchange (34)
         Next payload: Nonce (40)
          0... .... = Critical Bit: Not Critical
          Payload length: 104
          DH Group #: Default 768-bit MODP group (1)
          Key Exchange Data:
dcfae25205d1afcbc1b9d9a77464ad375229a747fc48e56621608e7533d617ce2a4b4f54ff3702f
0632c17ac7e8fa8ee1c52e4d0825bd5ff1d26e0197b106dae63bff8183fab5fcf9fe2bbe5ab3fbb
a37f26be4cda595139e894fb01d5d6f6c5
```

The following RFCs detail the use of various Diffie-Hellman groups, including the parameters to be used:

RFC3526 More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)

RFC5114 Additional Diffie-Hellman Groups for Use with IETF Standards.

Public Key Infrastructure

Public Key Infrastructure (PKI) is an architecture that manages the relationship between encryption keys and a related identity. Entities (devices or users) enroll into the PKI, where their public key is cryptographically bound to their identity.

A PKI allows hosts to communicate securely by exchanging certificates and authenticating themselves to each other. The benefit of using a PKI over a shared secret is that anyone or anything within a PKI can communicate without knowing the peers' shared secret. This is contrary to symmetric encryption.

The following components are used within a PKI.

Public Key Cryptography

Public key signing was originally invented to solve the key distribution problem and to provide authenticity.

Public-key cryptography, also called asymmetric cryptography, is a form of cryptography based on algorithms that require two separate keys, one of which is secret (or private) and one of which is public. Both the public and private keys are mathematically linked, which gives the ability to prove what is done with one key can be undone with the other.

Public key cryptography has many advantages over symmetric key cryptography: the primary one being that both parties are not required to know each other before communication can take place; instead, they all trust a common entity that signs certificates (containing an identity and public key), which is known as a chain of trust. The use of digital signatures enables the holder of a certificate and associated private key to prove their authenticity.

Certificate Authorities

A certificate authority (CA) has the function of managing certificate requests and issuing certificates to participating entities such as hosts, network devices, or users. The CA provides centralized key management, which all participating entities trust.

To validate the digital signature of the certificate authority, the receiver must first know the CA's public key. Typically this process is handled out of band or through an operation that is performed at installation. On Cisco IOS, you must first configure a trust point, which is a cryptographic container; you can then authenticate the certificate authority. This process involves obtaining a copy of the CA's certificate and validating the fingerprint (or signature) of it.

Digital Certificates

A digital certificate will link the digital signature to an identity. A digital certificate contains information to identify something, such as a name, serial number, organization, state, or IP address. It also contains a copy of the entity's public key. The CA that signs the certificate is an entity that all parties trust and that the receiver explicitly trusts to validate identities and to create digital certificates.

Digital Signatures Used in IKEv2

In the IKEv2 second message exchange (IKE_AUTH), both peers authenticate themselves. When certificates are used, the method of authentication is either using Rivest-Shamir-Adleman Signature (RSA-sig), or Elliptic-Curve Digital Signature Algorithm (ECDSA-sig).

ECDSA signatures are smaller than RSA signatures, and ECDSA keys require less bandwidth; the use of ECDSA-sig also has advantages of computational speed and efficiency when compared to RSA-sig.

The actual method by which authentication is proven has a peer signing a block of data (called signed octets) that contains data such as the IKE identity, nonce (number used once), and a value that is derived from the SKEY (SK_pr or SK_pi). The nonce is a random value that is used as an input into cryptographic calculations to add randomness.

As the IKE identity and nonce are included in the generation of key material, this will keep an malicious observer from performing a man-in-the-middle (MITM) attack. In an MITM attack, an attacker creates a secure channel to each peer impersonating the true peer. As the signed octets contain values such as the nonce, it is computationally infeasible for attackers to guess this value prior to attempting to insert themselves into the exchange.

This block of data, which can be referred as a message, is signed by the private key, which is sent within the authentication payload described in detail in the next chapter.

The signature can be described as;

signature = sign(message, private_key)

This signature can be validated by the receiving peer, by using the public key corresponding to the private key, which has a cryptographic relationship between both:

verify (message, signature, public_key) = pass | fail

The receiving host can verify that the message was signed by the holder of the private key by verifying the data contained in the AUTH payload. This is the case when RSA-sig or ECDSA-sig is

used as the authentication method. The IKE_AUTH exchange usually includes the certificate containing the public key with the cryptographic relationship to the private key that is used to sign the data. There is a case where the certificate used for authentication is not sent within the exchange; however it is also possible to send an SHA-1 hash of the certificate and a URL indicating the receiving host can obtain this; this is known a HASH and URL or HTTP-URL-Lookup.

Pre-Shared-Keys, or Shared Secret

IKEv2 gives clients the ability to authenticate using a shared secret. This secret, unlike digital certificates, must be communicated out of band. The shared secret is an ASCII or hexadecimal string of characters and is used when authenticating a device.

The AUTH payload consisting of the following:

AUTH = prf(prf(Shared Secret, “Key Pad for IKEv2”),<SignedOctets>)

It is worth noting the following problem. When a shared secret is used, if the initiator is tricked into connecting to a rogue VPN headend or if an MITM masquerades as the true responder, the AUTH payload is protected with the cryptographic material exchanged in the IKE_SA_INIT exchange. However, if an attacker has spoofed a response in which the initiator sends the authentication data, the attacker will be able to decrypt this and thus know all attributes used to generate the AUTH payload, apart from the shared secret. An attacker would be able to capture the AUTH payload and perform an offline brute-force attack, entering the correct (known) data for <SignedOctets> and the string “Key Pad for IKEv2,” while substituting values for the shared secret (for example, using a dictionary or brute force attack) until the value generated matched the value received. If the shared secret was a weak value using a dictionary word or a string with little entropy, it would be easily breakable with current computing power.

Encryption and Authentication

After the IKEv2 initial handshake, all messages are encrypted and authenticated. With regards to protected messages, IKEv2 implicitly uses Encapsulation Security Payload to protect the confidentially and integrity of IKE traffic. As there is no discrepancy for the type of security protocol required, no negotiation takes place.

As IPsec can use Encapsulation Security Payload, or Authentication Header, or both protocols, there is a requirement to negotiate this before the IPsec Security Association is created. This negotiation occurs within the second IKE exchange, known as IKE_AUTH.

Figure 1-1 illustrates the IKEv2 Encrypted payload format, where the use of Encapsulation Security Payload can be seen. It should be noted that this differs from the Encapsulation Security Payload header as described in RFC4303; there is no SPI, although this is contained within the IKEv2 header itself.

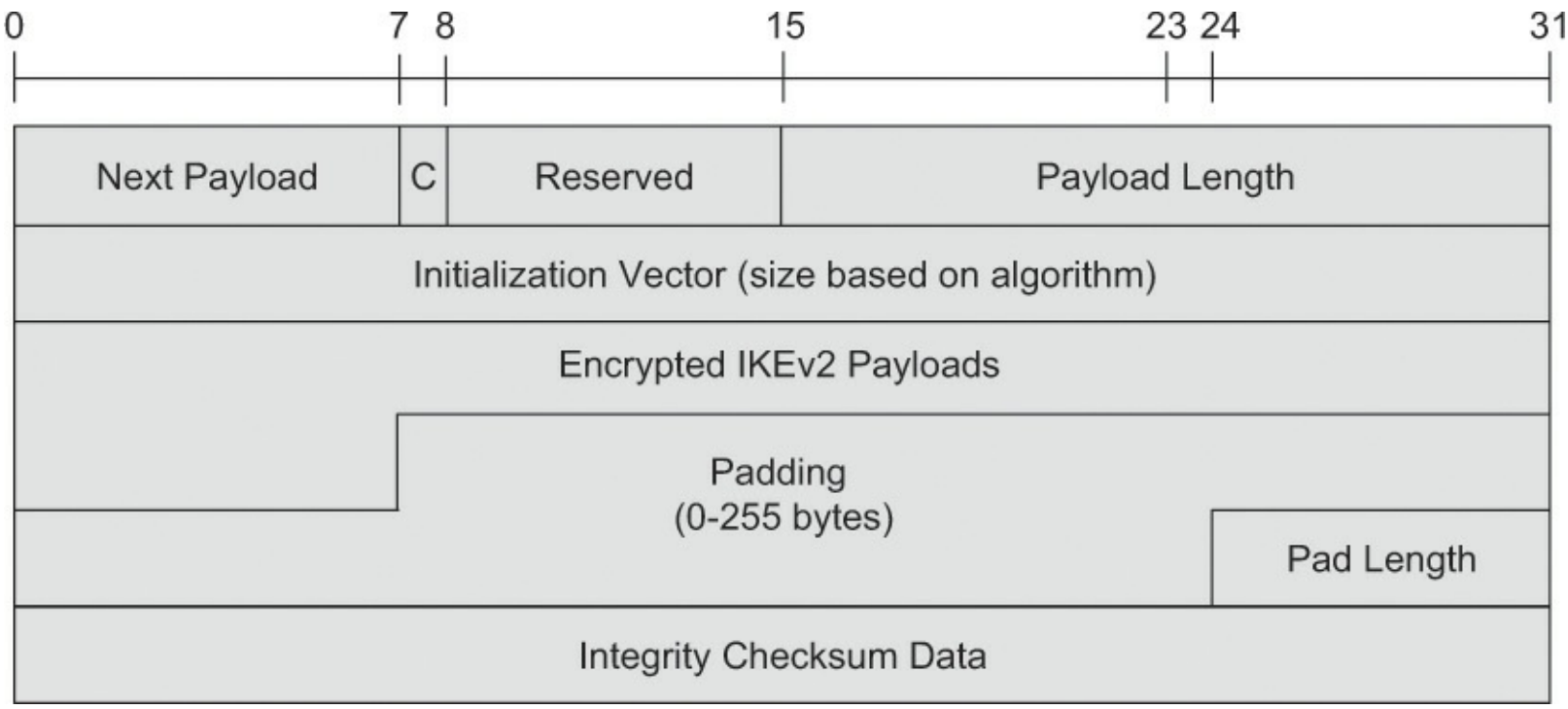


Figure 1-1 *IKEv2 Encrypted Payload Format*

The following illustrates a Proposal being negotiated by IKEv2 that will use Encapsulation Security Payload, which is Security Protocol ID 3 (with Authentication Header being ID 4).

Click here to view code image

```
Type Payload: Security Association (33)
                    Next payload: Traffic Selector - Initiator (44)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 144
                    Type Payload: Proposal (2) # 1
                        Next payload: Proposal (2)
                        0... .... = Critical Bit: Not Critical
                        Payload length: 40
                        Proposal number: 1
                        Protocol ID: ESP (3)
                        SPI Size: 4
                        Proposal transforms: 3
                        SPI: cf1346e0
```

Each protocol is covered in the sections that follow.

IP Authentication Header

The IP Authentication Header as described in RFC4302 is used to provide connectionless integrity and data origin authentication for IP datagrams and to provide protection against replay attacks. Note that this does **not** offer any confidentially services, and it will prevent only manipulation of traffic sent.

Data origin authentication and connectionless integrity are joint services, which can be commonly referred to as integrity, which is provided by the use of the Integrity Check Value (ICV) field within the Authentication Header, which contains keyed Message Authentication Codes, also known as cryptographic keyed hash. The ICV is generated by taking the message as an input along with a secret key and producing a fixed length hash output. This hash will be the value of the ICV and is determined by the integrity algorithm used.

The Authentication Header uses IP protocol 51; therefore the protocol header (IPv4, IPv6, or IPv6 Extension) immediately before the Authentication Header header will contain the value 51 in its Protocol (IPv4) or Next Header (IPv6, Extension) field.

With the Authentication Header, the whole of the header (except for any mutable fields) is authenticated, along with all payload data.

> **Note**
>
> For any mutable fields, the value is set to zero for purposes of the ICV computation. This ensures that IP checksum or IP TTL recalculations do not cause the receiver to incorrectly calculate the ICV.

Figure 1-2 illustrates both transport mode and tunnel mode when using Authentication Header with generic routing encapsulation (GRE). In all cases the outer IP header and subsequent data (including the Authentication Header) is authenticated. The modes of IPsec and GRE are described in detail in chapter 4, "IOS IPsec Implementation."

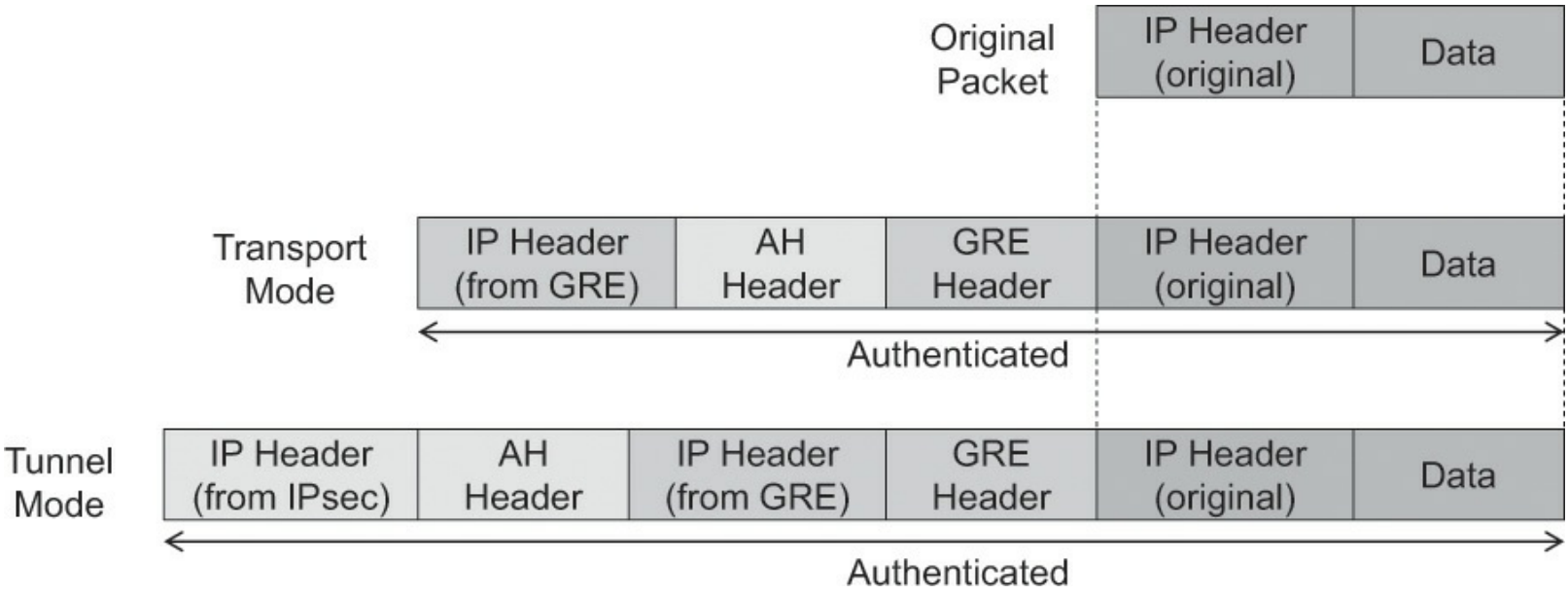


Figure 1-2 *Authentication Used With the Authentication Header*

> **Note**
>
> GRE will be covered in detail within chapter 4, "IOS IPsec Implementation."

As all fields in the packet headers are authenticated, if the outer IP address field changes, as would happen if using Network Address Translation (NAT) or Port Address Translation (PAT), then the receiving host would fail the authentication check when recalculating the ICV. For this reason Authentication Header cannot be used when NAT or PAT is used and is rarely used in customer deployments. As an alternative to using Authentication Header, Encapsulation Security Payload can be used with a null encryption (Encapsulation Security Payload-NULL). In this case only authentication and anti-replay services are provided by Encapsulation Security Payload, with no encryption service.

Figure 1-3 illustrates the Authentication Header format.

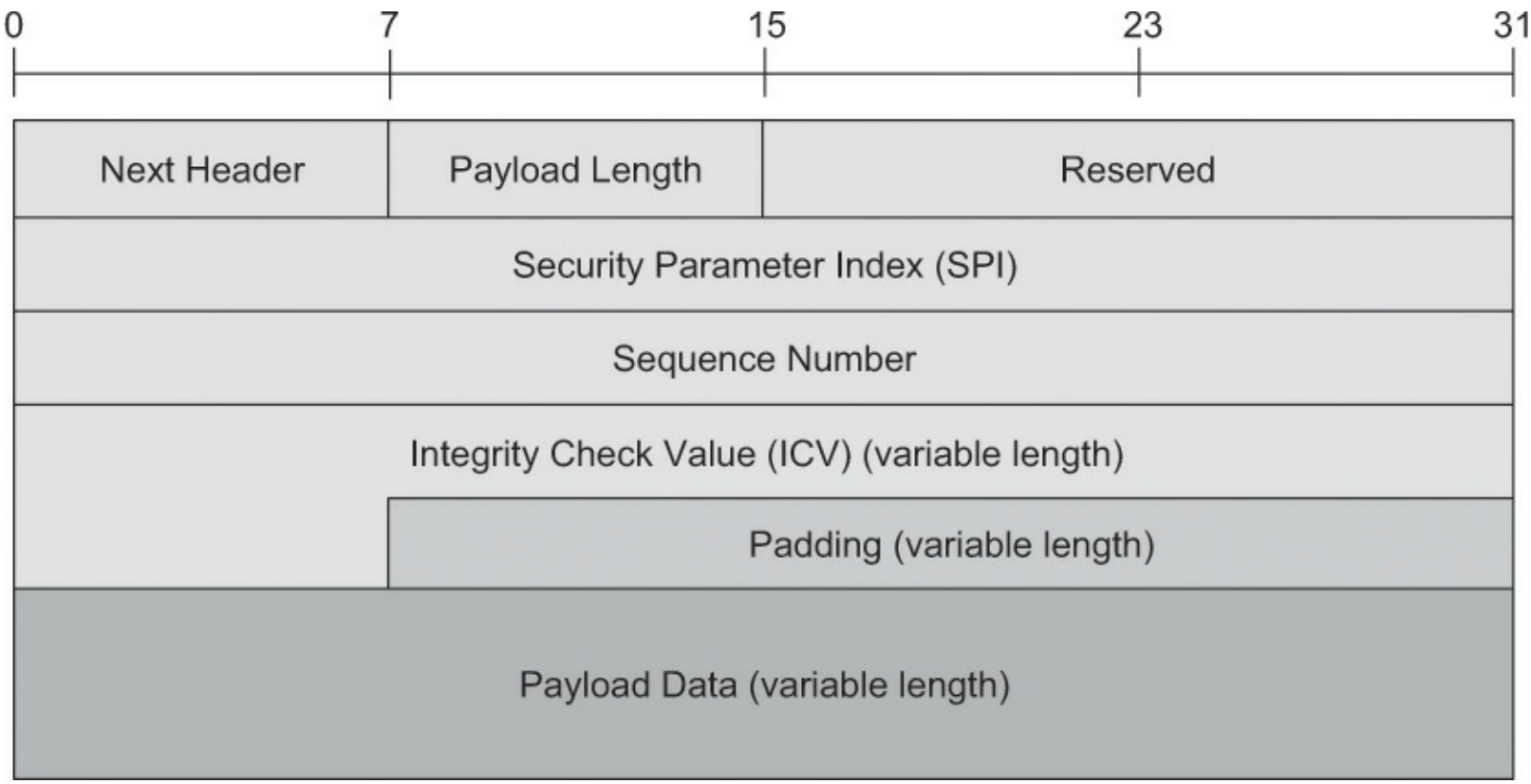


Figure 1-3 *Authentication Header Format*

Anti-Replay

Anti-replay functionality is provided by the use of sequence numbers, with every packet having a unique sequence number, along with an associated sliding window. The sender will set the initial sequence number to 1 and increment this number for all packets sent. Any packets received that were already received and correctly decrypted will be discarded, along with any packets that are received outside of the sliding window. This prevents an attacker taking copies of packets on the service provider network and replaying these, because without the anti-replay check they would be correctly decrypted.

The use of the anti-replay window mitigates the situation in which a malicious observer captures traffic and then replays it. If these checks were not implemented, the replayed traffic would be correctly decrypted and reach the intended destination. This traffic could cause congestion on links and disruption on the receiver.

On most networks today, there will be diverse routes, devices with bundled interfaces (port-channels), queuing mechanisms, and devices that process packets by parallel means (multiple CPUs or cores servicing a packet queue). These have the potential to make packets that were sent sequentially be received out of order.

The IPsec sliding window specifies the number of packets that can be received and be processed by the receiver that arrive late. The window begins at the sequence number for the last packet that was received and correctly decrypted (for Encapsulation Security Payload) or authenticated (when using Authentication Header). This value is known as X. If the next packet's sequence number is above the previous received sequence number or within X minus the sliding window value, then the packet will be processed.

As an example, if the sliding window was 100 packets and a packet with sequence number 522 was received, only packets with a sequence number 422 or higher would be processed.

IP Encapsulating Security Payload (ESP)

Encapsulating Security Payload as described in RFC4303 can be used to provide confidentiality, data origin authentication, connectionless integrity, and anti-replay services to traffic. Encapsulation Security Payload is the de-facto security protocol used by IPsec because it offers encryption, whereas Authentication Header does not.

Encapsulation Security Payload uses IP protocol 50, and the protocol header (IPv4, IPv6, or IPv6 Extension) immediately prior to the Encapsulation Security Payload header will contain the value 50 in the Protocol (IPv4) or Next Header (IPv6, Extension) field.

Figure 1-4 illustrates the Encapsulation Security Payload header format.

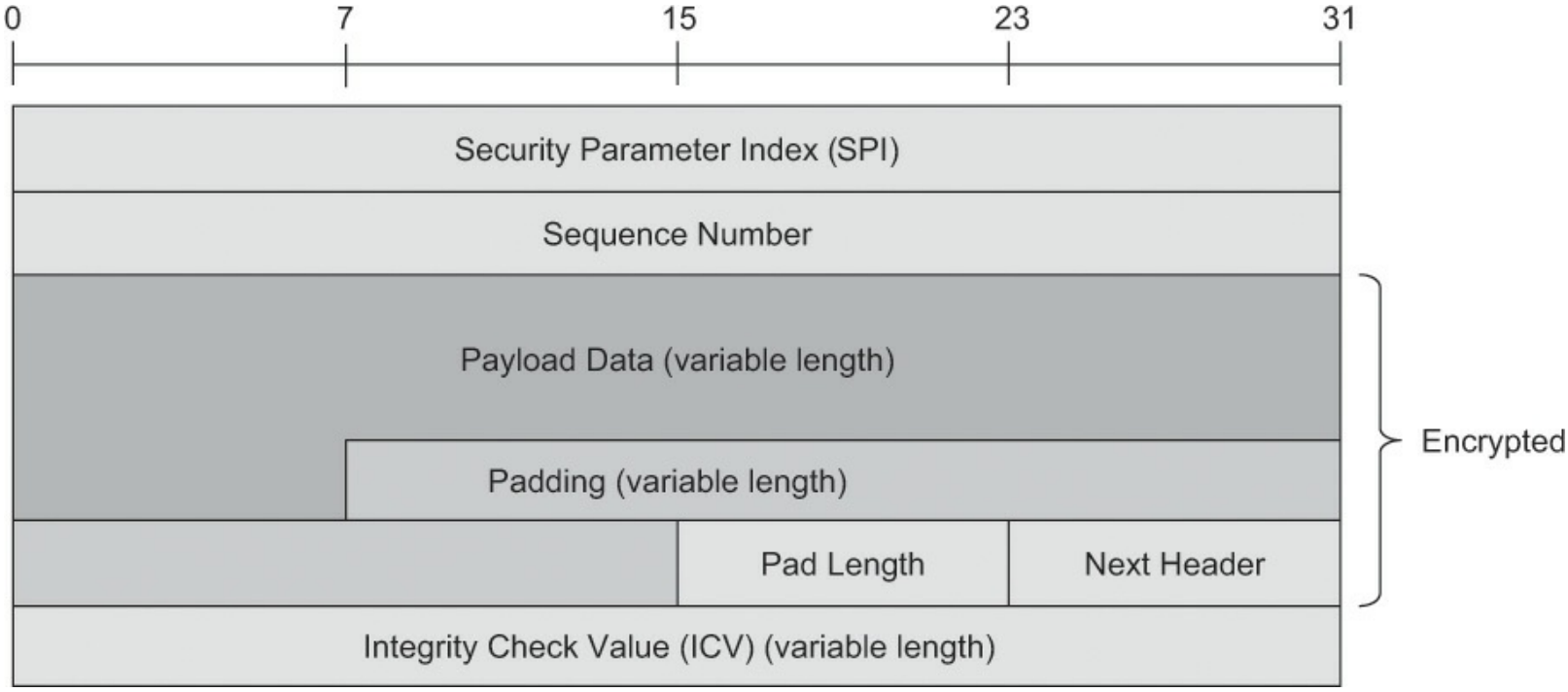


Figure 1-4 *Encapsulation Security Payload Header Format*

Authentication

Just as in Authentication Header, data origin authentication and connectionless integrity are joint services, which can be commonly referred to as integrity. This is provided by the use of the integrity check value (ICV) field within the Encapsulation Security Payload trailer, which contains keyed message authentication codes, also known as cryptographic keyed hash or hashed message authentication code (HMAC). The ICV is generated by taking the message as an input along with a secret key and producing a fixed length hash output. This hash will be the value of the ICV and is determined by the integrity algorithm used.

Encryption

Encapsulated Security Payload performs encryption for all traffic. All traffic being encapsulated inside of the Encapsulation Security Payload payload being encrypted. Traffic is encrypted using a secret, along with a random value (as the same message should never be encrypted with the same key), which is the initialization vector (IV).

Note

The Encapsulation Security Payload header itself is not encrypted, because it contains data such as the sequence number that is needed prior to decryption (for anti-reply checks) and the IV, which is required as an input to the decryption function.

Anti-Replay

Within Encapsulation Security Payload, anti-replay functionality is provided by the use of sequence numbers, with every packet having a unique sequence number, along with an associated sliding window. The sender will set the initial sequence number to 1 and increment the number with each packet. Any packets received that were already received and correctly decrypted will be discarded, along with any packets that are received outside of the sliding window. This prevents an attacker taking copies of packets on the service provider network and replaying these, as without the anti-replay check they would be correctly decrypted.

Encapsulation Security Payload Datagram Format

Figure 1-5 illustrates both transport mode and tunnel mode when Encapsulation Security Payload is used with GRE. In all cases the Encapsulation Security Payload Header and all subsequent data is authenticated and the original packet is always encrypted.

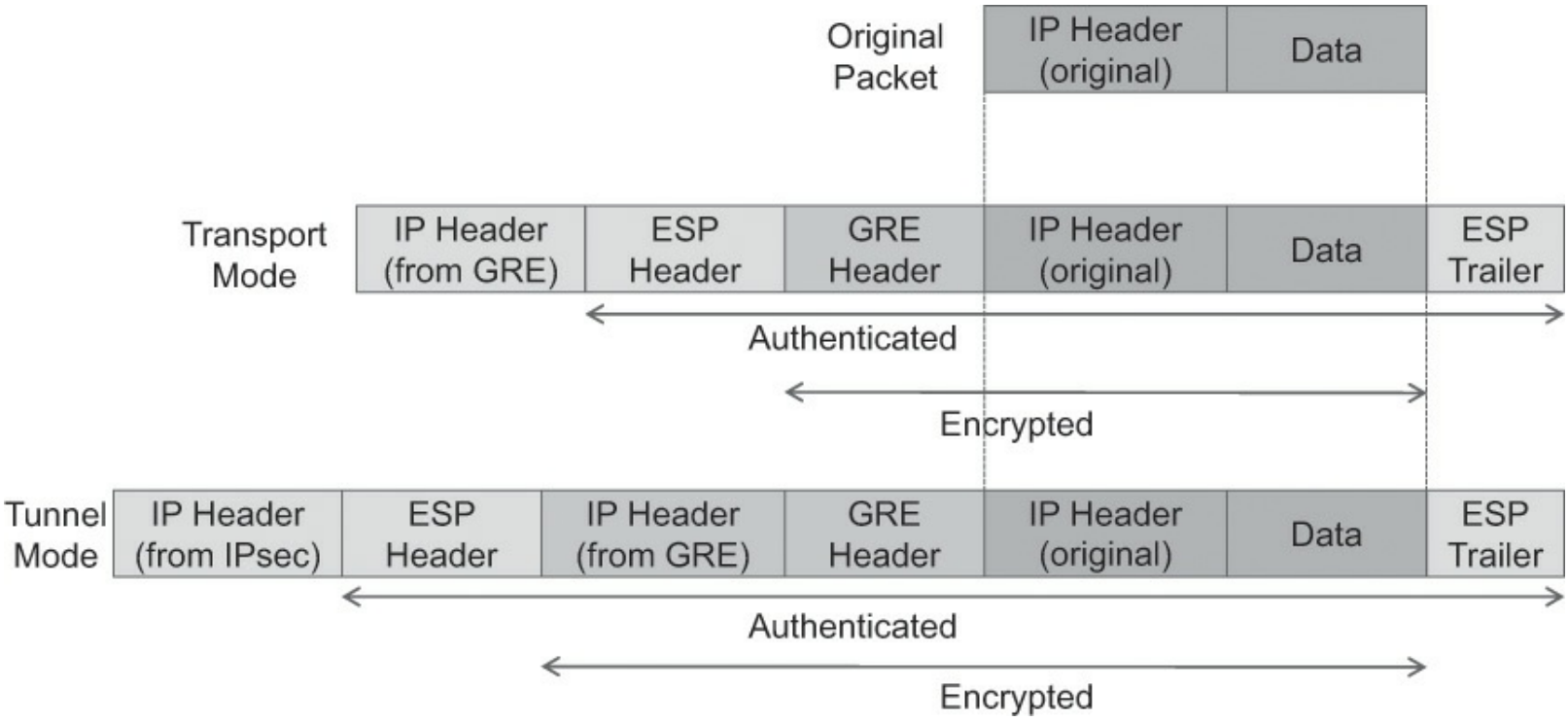


Figure 1-5 *Authentication and Encryption when Using Encapsulation Security Payload with GRE*

Unlike the Authentication Header, in which all fields in the packet headers are authenticated, Encapsulation Security Payload will not authenticate the outer IP header, which allows for any field to change the header without affecting the ICV, as would happen if Network Address Translation (NAT) or Port Address Translation (PAT) was in use. This results in Encapsulation Security Payload being compatible with NAT or PAT.

Encapsulating Security Payload Version 3

Encapsulating Security Payload version 3, which is described in RFC4303, brings a number of additions to the Encapsulation Security Payload protocol.

Extended Sequence Numbers

By default IPsec packets have 32-bit sequence numbers, with each packet having a unique sequence number. Rekey is mandatory before a sequence number rollover, which happens after 2^{32}–1 or 4,294,967,295 packets. With a constant traffic stream over an IPsec connection, an IPsec rekey would be required every six minutes for a 10Mbps link.

Encapsulation Security Payload version 3 allows for 64-bit sequence numbers, also known as extended sequence numbers (ESN) that increases the number of packets sent before a rekey will occur to 2^{64}–1 or 18,446,744,073,709,551,615 packets; this provides traffic to flow constantly through an IPsec protected 10Gbps interface for 100 years without a rekey.

The ability to use ESN is negotiated when the IPsec Security Association is negotiated.

> **Note**
>
> Currently IOS and IOS-XE does not support ESN.

Traffic Flow Confidentiality

By default, the padding field of the Encapsulation Security Payload header is limited to 255 bytes in length. Padding all packets by this size will generally be adequate to hide all traffic characteristics. Encapsulation Security Payload version 3 provides an optional field, which comes after the end of the encrypted data and before the beginning of the padding field, that can be used to contain additional padding that will be discarded by the receiver. This allows the size of the encrypted packet to be increased up to the maximum packet size when it is sent by the IPsec device.

Dummy Packets

Encapsulation Security Payload version 3 allows for "dummy" packets to be sent over the IPsec Security Association. These dummy packets can be inserted at random intervals to mask the absence of actual traffic or can be used to shape the encrypted traffic to match a required distribution. The next header field within the Encapsulation Security Payload header is used to indicate the payload type: a value of 4 indicates IPv4, a value of 41 indicates IPv6, and a value of 6 indicates TCP. In the case of dummy packets, the next header is set to a value of 59, which indicates to the receiving host that there is no next header and this should be discarded. Obviously the next header field is encrypted within the Encapsulation Security Payload header and can only be validated by the receiver.

Modes of IPsec

Authentication Header or Encapsulation Security Payload can operate in either of two modes: transport mode or tunnel mode. In transport mode protection is provided primarily for next layer protocols. By this we mean that plaintext packets are protected by the said protocol. Tunnel mode provides encapsulation for the plaintext traffic, prior to encryption.

IPsec Transport Mode

With IPsec transport mode, a plaintext IPv4 or IPv6 packet is protected by IPsec for confidentiality and/or integrity protection.

The IPsec transport mode overhead compared to the original packet is just calculated as the crypto overhead (see chapter 15, "Calculating IPsec Overhead and Fragmentation").

Figure 1-6 shows the encapsulation of a UDP over IPv4 packet into Encapsulation Security Payload transport mode with IPv4 transport. The UDP traffic endpoints are 10.x.x.x, with the packet being encrypted, and when the Encapsulation Security Payload header is inserted, it comes after the original IP header. The protocol field in the outer IP header is set to 50 (0x32) which denotes the next header is Encapsulation Security Payload.

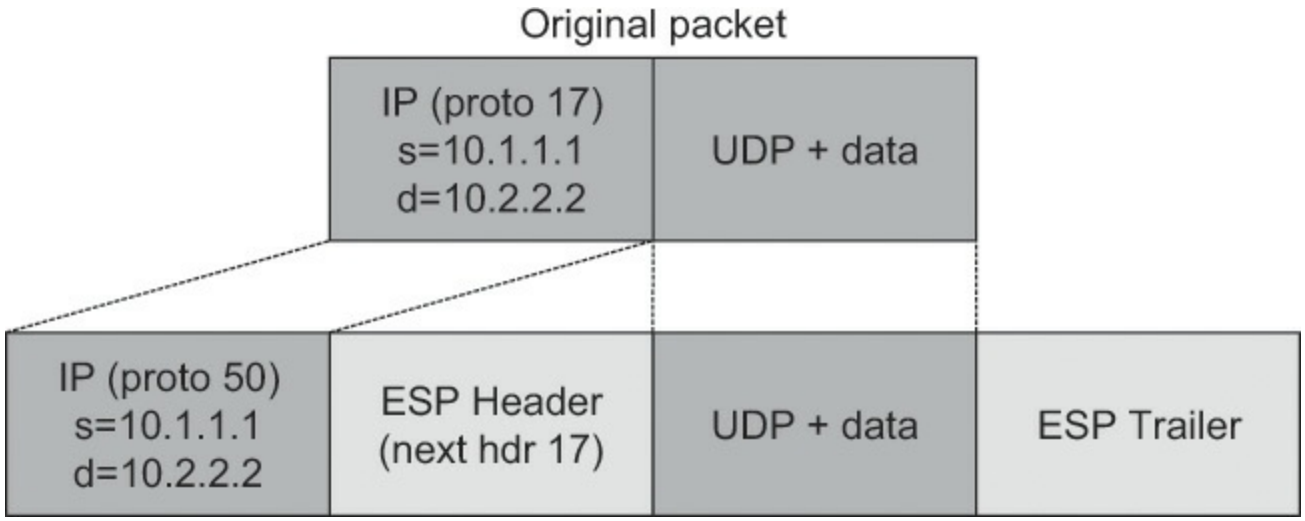


Figure 1-6 *Before and After IPsec Transport Mode*

This type of protection would be possible only with crypto maps, where traffic is protected when passing through an interface, and not with tunnel interfaces (which are covered in chapter 4, "IOS IPsec implementation"), because if a tunnel interface was configured for pure IPsec and transport mode was used, the IP header of the protected packet would be used after Encapsulation Security Payload protection. This is considered a misconfiguration, and at the time of this writing, Cisco IOS actually would autocorrect this to use tunnel mode instead of transport mode if a transform set of type mode transport is used without the additional required keyword.

IPsec Tunnel Mode

With IPsec tunnel mode, a plaintext IPv4 or IPv6 packet is encapsulated into another IPv4 or IPv6 packet protected by IPsec for confidentiality and/or integrity protection.

The IPsec tunnel mode overhead compared to the original packet is:

- 20 bytes (more if IP options are present) for the outer IPv4 header, or
- 40 bytes (more if extension headers are present) for the outer IPv6 header, plus the crypto overhead (see chapter 15, "Calculating IPsec Overhead and Fragmentation").

Figure 1-7 shows the encapsulation of a UDP over IPv4 packet into Encapsulation Security Payload tunnel mode with IPv4 transport. The UDP traffic endpoints are 10.x.x.x, and the IPsec tunnel endpoints are 172.16.x.x. The protocol field in the outer IP header is set to 50 (0x32), which denotes the next header is Encapsulation Security Payload.

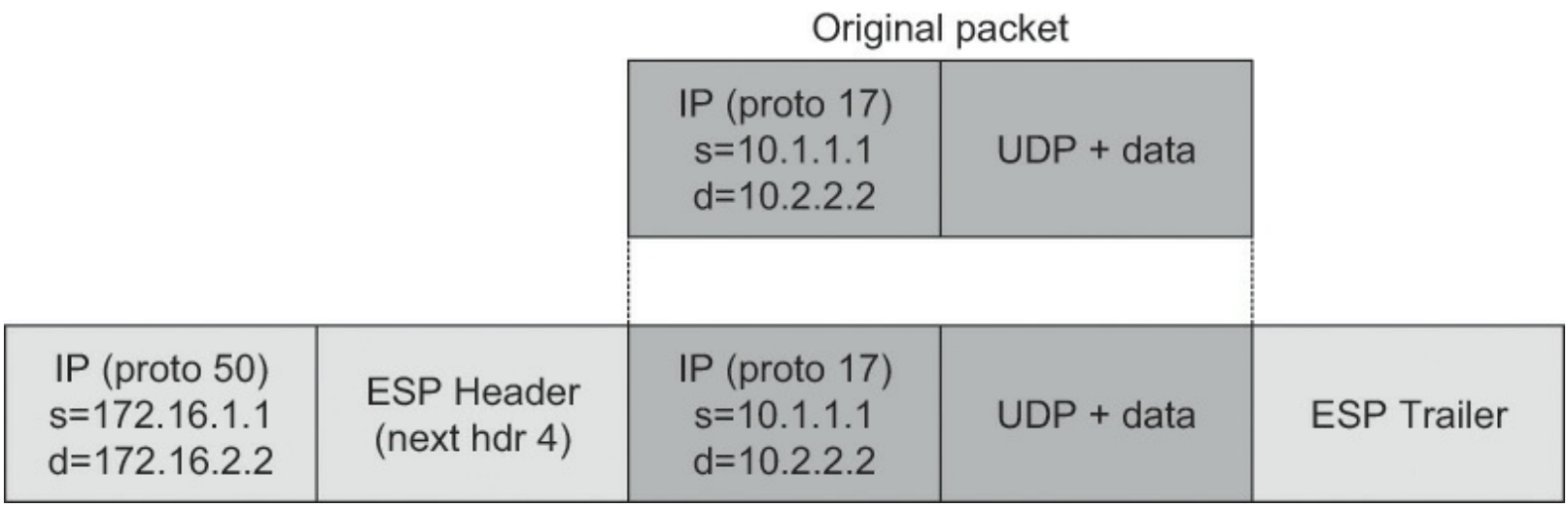


Figure 1-7 *Before and After IPsec Tunnel Mode*

Summary

This chapter introduced the IPsec protocol and the components that make up IPsec. As you are now aware, IPsec has the ability to use Authentication Header or Encapsulation Security Payload to provide security services to protect the IPsec Security Association. Each security service was described, and you will now be familiar with what each delivers.

The Internet Key Exchange was introduced within this chapter; although IKE is the protocol used to create and maintain IPsec Security Associations, IPsec itself is a separate entity.

The chapter discussed the databases used by IPsec and the ways these integrate together; the Security Association Database defines what is protected, and the Security Policy Database details how traffic is protected.

Encapsulation Security Payload version 3 introduces a number of security features, which include traffic flow confidentiality, "dummy" packets, and extended sequence numbers.

Tunnel mode allows for traffic to be protected within the IPsec Security Association, which performs limited traffic flow confidentiality by hiding the internal IP address, allowing for traffic to be tunneled within the IPsec Security Association. However, it requires an additional IP header. Transport mode requires that the protected traffic's IP header is used to transport the packet on the wire, but it results in less overhead than Tunnel mode.

References

http://www.cisco.com/web/about/security/intelligence/nextgen_crypto.html Next Generation Encryption

https://tools.ietf.org/html/rfc4301 Security Architecture for the Internet Protocol

https://tools.ietf.org/html/rfc4302 IP Authentication Header

https://tools.ietf.org/html/rfc4303 IP Encapsulating Security Payload

https://tools.ietf.org/html/rfc4304 Extended Sequence Number (ESN)

Part II: Understanding IKEv2

Chapter 2. IKEv2: The Protocol

This chapter takes you through the lifecycle of the Internet Key Exchange version 2. The chapter is structured to provide an overview of IKEv2, then detailed information about each exchange type (IKE_SA_INIT, IKE_AUTH, CREATE_CHILD_SA and INFORMATIONAL), so you will have a deep understanding of what is occurring within each.

Understanding the protocol will assist engineers when deploying and troubleshooting the technology, as well as designers making decisions that work when scoping VPN architectures.

This chapter was designed to be accessible, yet also still maintain details from the IKEv2 RFC and minimize output from Cisco-specific technologies to make this relevant if you work in a vendor agnostic environment.

Outputs are taken from packet captures saved in text format. The intent is to show clearly the packet structures along with the data being passed. In some cases the packet was encrypted, and it had to be decrypted to obtain this information.

IKEv2 Overview

The Internet Key Exchange Protocol Version 2 (IKEv2) is described in RFC7296. A Request for Comments (RFC) is a publication created by the Internet Engineering Task Force (IETF) and the Internet Society. The IETF is the primary technical body that develops and sets standards for use on the Internet.

An RFC is published in the form of a memorandum describing the methods, behaviors, and/or innovations applicable to the working of the Internet and Internet-connected systems. Before an RFC is published, it goes through a process in which it is peer reviewed and analyzed by members of the IETF. Before RFC7296 was published, the IETF published RFC4306, which described IKEv2 and was published as an Internet standard in 2005. RFC4306 has since been updated and replaced by RFC5996, which was published in 2010. In October 2014, RFC5996 was updated and replaced by RFC7296, making RFC5996 obsolete.

IKEv2 consists of an initial handshake (known as IKE_SA_INIT or initial exchange) in which both parties negotiate a set of cryptographic algorithms, exchange Diffie-Hellman public values, and establish a shared secret that can then be used to derive additional key material to protect subsequent IKEv2 exchanges. Randomness is added to the exchange in the form of nonces (‘number used once’).

The second part of the exchange (known as IKE_AUTH or authentication) is secured using cryptographic algorithms and key material obtained from the initial handshake. Each party will exchange identities and provide authentication data to validate their identity, using an integrity check generated from the secret key associated with their identity.

Peers will exchange cryptographic algorithms that will be used to create one or more IPsec Security Association using Encapsulated Security Payload or Authentication Header.

Should additional pairs of IPsec Security Associations be required, then an additional IKEv2 exchange will occur to create these using a CREATE_CHILD_SA exchange. When IKEv2 is used, these additional IPsec Security Associations are known as Child SAs.

The IKEv2 protocol employs a simple denial of service (DoS) protection mechanism that mitigates blind spoofing attacks that could consume expensive resources.

Every IKEv2 message exchange consists of request/response pairs, with the sender of the request being responsible for retransmitting messages if it does not receive a reply.

On rekey of the IKEv2 SA, a new CREATE_CHILD_SA exchange will occur; this will include exchanging cryptographic algorithms that will be used to protect the new IKEv2 SA. A fresh Diffie-Hellman exchange will occur and be used to generate new key material to protect the subsequent IKEv2 SA that is created.

On rekey of an IPsec Security Association, a new CREATE_CHILD_SA exchange will occur, which consists exchanging cryptographic algorithms that will be used to protect the new IPsec Security Association. Optionally a fresh Diffie-Hellman exchange can occur that will be used in generating new key material to protect the subsequent IPsec Security Association. It should be noted that on a rekey no authentication occurs, RFC4478, Repeated Authentication in Internet Key Exchange (IKEv2) Protocol, describes how re-authentication can occur in IKEv2.

IKEv2 allows additional control messages to be exchanged once the IKEv2 SA is completed using an INFORMATIONAL exchange. These messages allow for actions such as liveness checking (also known as keepalives) and the deletion of SAs.

The IKEv2 Exchange

In IKEv2 terminology, the first phase consists of the (usually four) messages that create the IKEv2 SA and the first associated IPsec Security Association (known as a “child SA”).

The IKEv2 SA is a protected tunnel that allows for subsequent messages to be sent cryptographically protected; this allows the sending of encrypted and authenticated notification messages, reliable dead-peer detection, and creation of additional child SAs.

IKEv2 uses User Datagram Protocol (UDP) as a transport mechanism and uses ports 500 and 4500. All exchanges should begin with using UDP port 500 as the source and destination port. Although the RFC is not strict on the source port used, most implementations will use the recommended settings. Figure 2-1 illustrates the IKE_SA_INIT and IKE_AUTH exchanges with a very high-level explanation of the purpose of each.

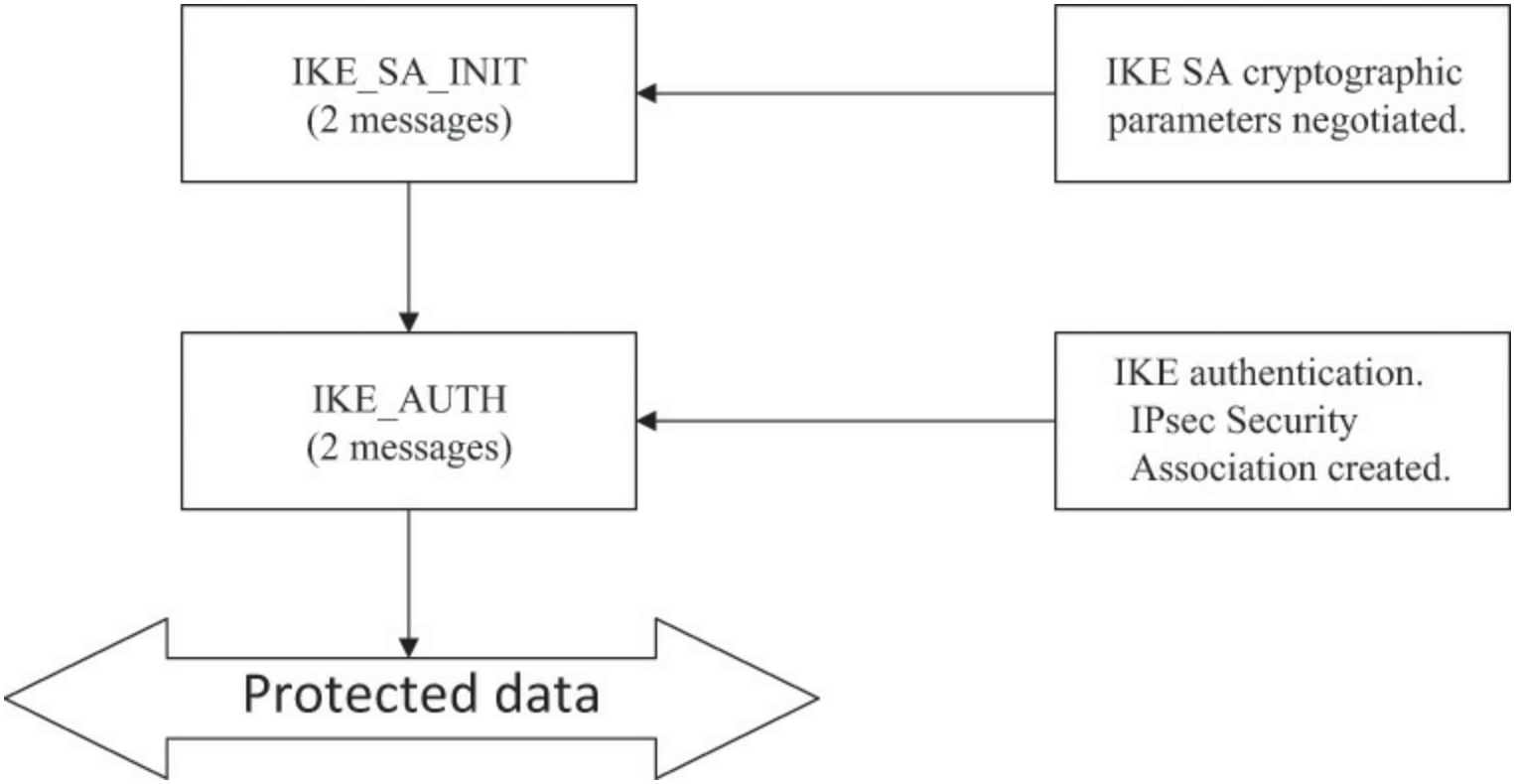


Figure 2-1 *IKEv2 Exchange Overview (RFC7296)*

IKE_SA_INIT

The IKE_SA_INIT is the initial IKEv2 exchange between both parties, which are known as the initiator and the responder, the initiator being the party that starts the IKEv2 session.

The initiating peer will send a proposal of cryptographic algorithms that it supports, which will be used to secure the IKE SA. A Diffie-Hellman public key value is sent, which is used to generate the shared secret. A nonce (number used once) is included in the exchange, which adds randomness into the key generation.

The responder, on receipt of the initial IKE_SA_INIT request sent by the initiator, will check that it supports at least one combination of the proposed cryptographic algorithms that are to be used to create the IKE SA. It will select the most preferred algorithms and reply with these.

The responder will check the Diffie-Hellman value received and, assuming that it supports this value, will reply with its own Diffie-Hellman public value.

When the responder has generated and sent the Diffie-Hellman parameters, it will have enough information to generate the secret keys that then protect the rest of the IKEv2 traffic.

When the initiator receives the responder's reply containing the Diffie-Hellman parameters, it will have enough information to generate the secret keys that then protect the rest of the IKEv2 traffic.

The IKE_SA_INIT packet contains a header consisting of Security Parameter Indexes (SPIs), version number, and flags.

NAT detection can be performed in the IKE_SA_INIT exchange; if this is required NAT_DETECTION payloads will be added in the initial exchange. Figure 2-2 illustrates the IKE_SA_INIT exchange.

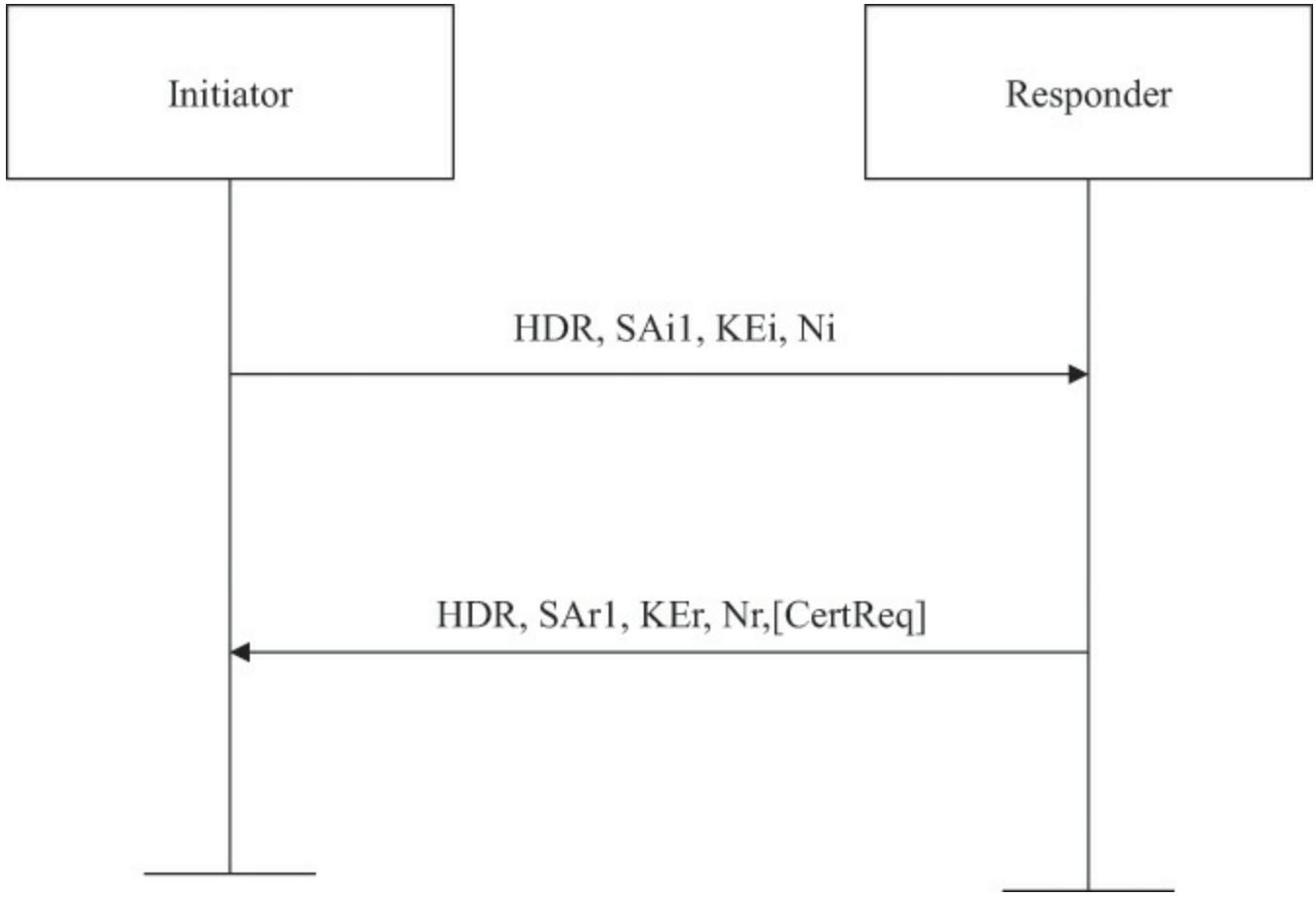


Figure 2-2 *IKE_SA_INIT Exchange*

The following describes each of the parameters of the IKE_SA_INIT exchange.

- **HDR:** IKE header
- **SAi1**: Sets of cryptographic algorithms proposed by initiator
- **SAr1**: Specific of cryptographic algorithms chosen by responder
- **KEi**: Initiator key exchange material (uses DH group with highest local priority)
- **KEr**: Responder key exchange material
- **Ni** : Initiator nonce
- **Nr**: Responder nonce
- **CertReq**: Certificate request (optional). The square brackets [ ] surrounding the CertReq payload denotes this as optional.

Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure medium. This shared secret key then becomes the input used to generate key material that secures the IKE SA.

For modular exponential Diffie-Hellman groups, one party will create its public value using the formula

$$X = g^x \bmod p$$

where

g generates a subgroup of size q (where q is a prime divisor of p-1)

x is a value between 1 and q-1

p is a prime

The peer will generated its own private value y and public value Y, using the formula.

$$Y = g^y \bmod p$$

The value x is generated by the device and this is kept privately. If an intruder was able to retrieve this, then the shared secret could be obtained and hence all encrypted IKEv2 packets could be decrypted. The g and p values are defined within the RFC for the relevant Diffie-Hellman group, for Modular Exponential groups, see RFC7296; for groups 1 and 2, RFC3526; for groups 5, 14, 15, and16, RFC5903; for groups 19, 20 and 21, RFC5903; and for group 24 RFC5114.

The public value exchanged is used to create a shared secret using the formula

$$s = Y^x \bmod p$$

Where

s = shared secret

x = local private value

Y = peer's public value

p = prime

It can be seen that $Y^x \bmod p$ equals $g^{xy} \bmod p$ equals $g^{yx} \bmod p$ equals $X^y \bmod p$. This allows each peer to derive the same shared secret by exchanging only their public values.

Assuming that each peer is using the same Diffie-Hellman group (which is defined within the RFC describing the group used), then each peer will know the values g and p and so can construct s, when the peer's public value (y) is received.

The initiator will generate a public Diffie-Hellman value and include this in the IKE_SA_INIT exchange, within a Key Exchange payload (denoted by KE in all diagrams). The Key Exchange payload consists of the IKE generic payload header followed by the Diffie-Hellman public value itself. The payload indicates which Diffie-Hellman group is used, and the size of the public value will correspond to this group; as a rule of thumb elliptic curve groups are smaller than modulus Diffie-Hellman groups, with some sizes being considerably different. For example, group 16, which uses a 4096-bit modulo Diffie-Hellman value, is sixteen times bigger than group 19, which uses a 256-bit elliptic curve value. However, elliptic curve groups give greater security for the small sizes and small work rate, so their use is recommended where possible.

The following example illustrates a Key Exchange payload that was taken from an IKE_SA_INIT exchange. The Diffie-Hellman group can be seen, along with the public value, which is contained within the key exchange data field.

Click here to view code image

```
Type Payload: Key Exchange (34)
        Next payload: Nonce (40)
        0... .... = Critical Bit: Not Critical
        Payload length: 200
        DH Group #: 1536 bit MODP group (5)
        Key Exchange Data: 973d1a029804cb52720701ce384322686c4f939aa78727e5...
```

The initiator is really guessing the Diffie-Hellman group that is used on the responder; the value is

just sent and not negotiated. For efficiency, the initiator should use a Diffie-Hellman value that it knows that the responder will accept; otherwise expensive cryptographic calculations are performed that can be wasted. On receipt of the initiator's IKE_SA_INIT request, the responder will either accept or reject the value, depending on whether the group used by the initiator is configured locally.

Taking all other factors into consideration and assuming that the initiator sent a value that is acceptable to the responder, then the responder will generate its own Diffie-Hellman public value and reply with this. It is common for a number of values to be precalculated so that expensive cryptographic processing doesn't occur when an initial request is received. It is also common for implementations to reuse values between connections. At this stage the responder will be able to generate the secret key from the initiator's public Diffie-Hellman value and its own private Diffie-Hellman value.

The initiator will include the Diffie-Hellman groups that it supports within the Security Association payload; this allows the responder to select another group should the initiator's presented public value not be supported.

If the initiator guessed wrong and the responder didn't accept the proposed Diffie-Hellman value, the latter will respond with a Notify payload of type INVALID_KE_PAYLOAD, indicating the selected group that it wishes to use.

In this case, the initiator must retry the IKE_SA_INIT request with the corrected Diffie-Hellman group (assuming that this group is supported by the initiator). The initiator must again propose its full set of acceptable cryptographic suites, because the rejection message was unauthenticated; otherwise an active attacker could trick the endpoints into negotiating a weaker suite than a stronger one that they both prefer.

Figure 2-3 shows the IKE_SA_INIT exchange increasing from two packets to four when an initiator includes a Diffie-Hellman group that the responder does not support. The second request from the Initiator contains a Diffie-Hellman group that the responder has advised that it supports in the INVALID_KE_PAYLOAD.

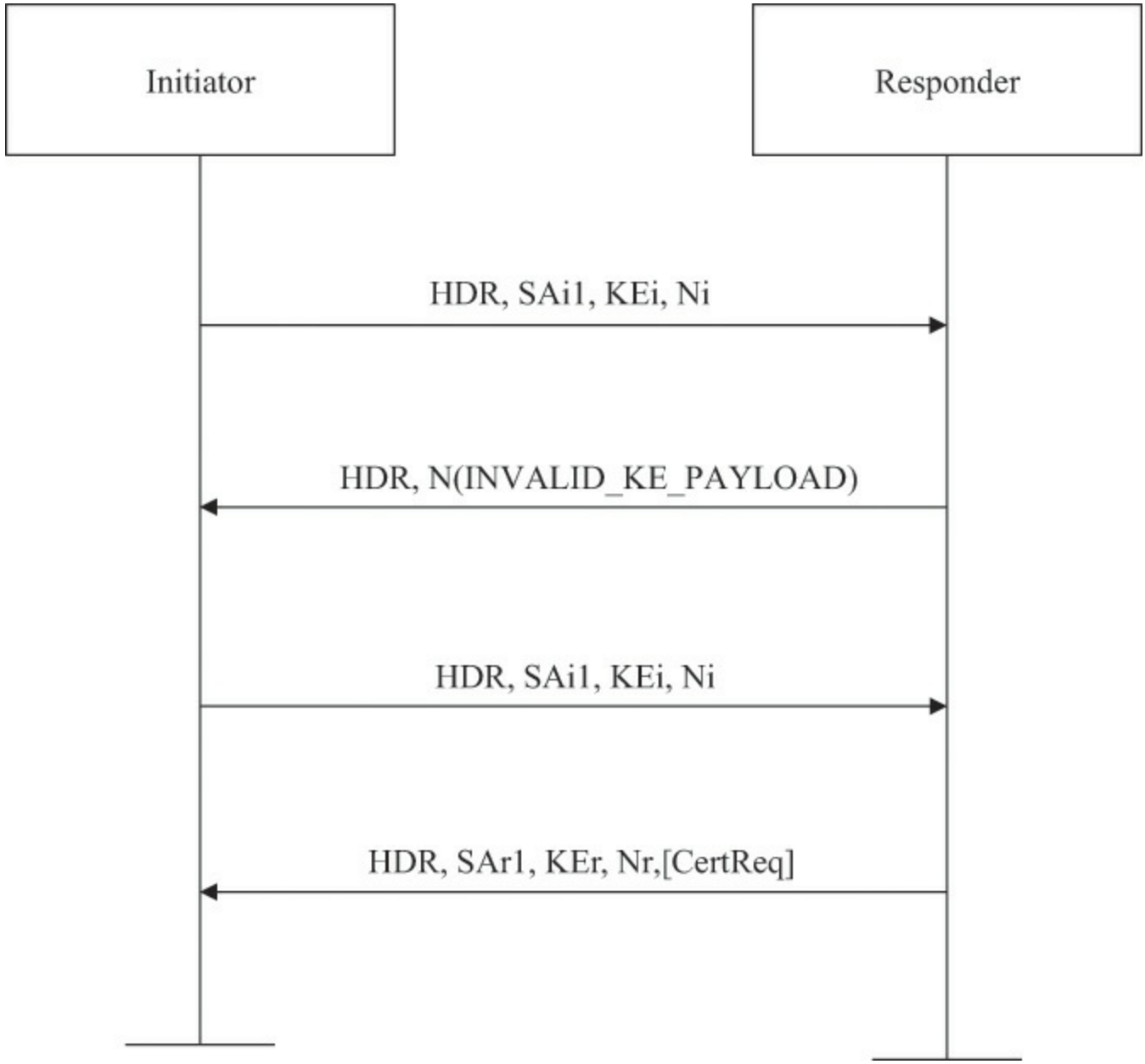


Figure 2-3 *IKE_SA_INIT with Failed Diffie-Hellman Group*

Additional tests are performed by some implementations (for example, Cisco IOS), when the implementation uses the checks proposed by RFC6989, Additional Diffie-Hellman Tests for the Internet Key Exchange Protocol Version 2.

If an implementation reuses the same Diffie-Hellman values for multiple connections, then an intruder could send a malformed Diffie-Hellman public key value, which would then allow the attacker to efficiently determine the Diffie-Hellman private value the peers used. RFC6989 actually proposes that the check be carried out regardless of whether the same Diffie-Hellman private values are reused.

When a device that supports these checks receives an IKE_SA_INIT request containing an elliptic curve public value that it supports, the host will perform a check on the received value to validate that it is not erroneous. Should the received value fail the check, then this public Diffie-Hellman value should not be used, and an INVALID_SYNTAX error notification should be sent to the peer to indicate that the received value was faulty.

Security Association Proposals

Within the IKE_SA_INIT exchange, peers will negotiate cryptographic algorithms that are used to protect the subsequent IKEv2 payloads (IKE_AUTH exchange onwards). The algorithms that a peer supports will be sent within a Security Association payload (denoted by SA within documentation).

The Security Association payload will contain Proposals, which are groups of cryptographic algorithms defined within Transforms; the Transforms support attributes that are negotiated for the protocols used to build an IPsec Security Association, IKE, Authentication Header, and Encapsulation Security Payload.

Table 2-1 displays the types of Transforms along with the method used.

Description	Transform Type	Used In
Encryption Algorithm (ENCR)	1	IKE and Encapsulation Security Payload
Pseudorandom Function (PRF)	2	IKE
Integrity Algorithm (INTEG)	3	IKE, Authentication Header, optional in Encapsulation Security Payload
Diffie-Hellman group	4	IKE, optional in Encapsulation Security Payload and Authentication Header
Extended Sequence Numbers (ESN)	5	Authentication Header and Encapsulation Security Payload

Table 2-1 *Types of Transforms and Methods Used*

> **Note**
>
> If a combined mode cipher is used for IKE, such as AES-GCM, then an integrity algorithm is not required.

The cryptographic algorithms used for encryption and integrity contained within a transform consist of encryption, integrity, or combined mode ciphers. Combined-mode ciphers support both encryption and integrity within the same cipher method (AES-GCM as described in RFC4106 or AES-CCM as described in RFC4301).

A proposal will only contain the same mode of algorithms within the transform, so only encryption and integrity algorithms could coexist within a transform, never with a combined mode algorithm. If an initiator wants to propose both combined mode ciphers and mixed ciphers, then it must include two proposals. RFC7296 recommends using combined mode ciphers where possible.

If a peer proposes multiple proposals, it must send the preferred algorithms first.

A peer receiving one or more proposals will select the proposal with the transforms that it supports. If there are multiple transforms of the same transform types, for example transform type 1 (encryption algorithms) with transforms of 3DES, AES-CBC-128, AES-CBC-256, the receiving host can select any of these transforms (assuming that it supports them), which equates to a logical OR. If there are

multiple transforms with different transform types, the selection is a logical AND of the different transform types. So if the proposal also contained transform type 3 (integrity algorithms) with MD5 and SHA-1, the receiver would select one transform of transform type 1 AND one of transform type 3.

Table 2-2 shows an example of proposed attributes within a transform.

Encryption	Integrity	ESN Support
AES-CBC-256	AUTH_HMAC_MD5_96	Y
AES-CBC-128	AUTH_HMAC_SHA1_160	N
3DES	AUTH_HMAC_SHA2_384	

Table 2-2 *Example of Proposed Attributes within a Transform*

The peer can use any of the algorithms in a mix-and-match basis, so if a peer received a request containing the transforms listed within Table 2-2, it could choose any set of algorithms depending on what is supported locally; for example:

AES-CBC-256 with AUTH_HMAC_SHA1_160 and ESN support

or

3DES with AUTH_HMAC_MD5_96 and no ESN support

or

AES-CBC-256 with AUTH_HMAC_SHA2_384 and ESN support

Note that IKEv2 requires a PRF to be included within the transform, which is used in the generation of the secret key material. IPsec does not require this, so the only time a PRF would be seen within an IKEv2 transform is only when the peers are negotiating an IKEv2 SA and not an IPsec Security Association.

The Key Exchange payload in the IKE_SA_INIT exchange will include the Diffie-Hellman public value. In addition, any other Diffie-Hellman groups that are supported by the sending device are included within the Security Association proposal. This allows for the peer to select another Diffie-Hellman group if the received public value is not supported.

The following example illustrates a Security Association proposal header that displays the Security Association payload from an initiator of a IKE_SA_INIT exchange; each transform type can be seen, along with the transform ID that denotes the attribute proposed.

Click here to view code image

```
Type Payload: Security Association (33)
    Next payload: Key Exchange (34)
    0... .... = Critical Bit: Not Critical
    Payload length: 136
    Type Payload: Proposal (2) # 1
        Next payload: NONE / No Next Payload  (0)
        0... .... = Critical Bit: Not Critical
        Payload length: 132
        Proposal number: 1
        Protocol ID: IKE (1)
        SPI Size: 0
        Proposal transforms: 14
```

```
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 12
    Transform Type: Encryption Algorithm (ENCR) (1)
    Transform ID (ENCR): ENCR_AES_CBC (12)
    Transform IKE2 Attribute Type (t=14,l=2) Key-Length : 256
        1... .... .... .... = Transform IKE2 Format: Type/Value
        Transform IKE2 Attribute Type: Key-Length (14)
        Value: 0100
        Key Length: 256
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 12
    Transform Type: Encryption Algorithm (ENCR) (1)
    Transform ID (ENCR): ENCR_AES_CBC (12)
    Transform IKE2 Attribute Type (t=14,l=2) Key-Length : 192
        1... .... .... .... = Transform IKE2 Format: Type/Value
        Transform IKE2 Attribute Type: Key-Length (14)
        Value: 00c0
        Key Length: 192
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 12
    Transform Type: Encryption Algorithm (ENCR) (1)
    Transform ID (ENCR): ENCR_AES_CBC (12)
    Transform IKE2 Attribute Type (t=14,l=2) Key-Length : 128
        1... .... .... .... = Transform IKE2 Format: Type/Value
        Transform IKE2 Attribute Type: Key-Length (14)
        Value: 0080
        Key Length: 128
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 8
    Transform Type: Pseudo-random Function (PRF) (2)
    Transform ID (PRF): PRF_HMAC_SHA2_512 (7)
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 8
    Transform Type: Pseudo-random Function (PRF) (2)
    Transform ID (PRF): PRF_HMAC_SHA2_384 (6)
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 8
    Transform Type: Pseudo-random Function (PRF) (2)
    Transform ID (PRF): PRF_HMAC_SHA2_256 (5)
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 8
    Transform Type: Pseudo-random Function (PRF) (2)
    Transform ID (PRF): PRF_HMAC_SHA1 (2)
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 8
```

```
                Transform Type: Pseudo-random Function (PRF) (2)
                Transform ID (PRF): PRF_HMAC_MD5 (1)
            Type Payload: Transform (3)
                Next payload: Transform (3)
                0... .... = Critical Bit: Not Critical
                Payload length: 8
                Transform Type: Integrity Algorithm (INTEG) (3)
                Transform ID (INTEG): AUTH_HMAC_SHA2_512_256 (14)
            Type Payload: Transform (3)
                Next payload: Transform (3)
                0... .... = Critical Bit: Not Critical
                Payload length: 8
                Transform Type: Integrity Algorithm (INTEG) (3)
                Transform ID (INTEG): AUTH_HMAC_SHA2_384_192 (13)
            Type Payload: Transform (3)
                Next payload: Transform (3)
                0... .... = Critical Bit: Not Critical
                Payload length: 8
                Transform Type: Integrity Algorithm (INTEG) (3)
                Transform ID (INTEG): AUTH_HMAC_SHA2_256_128 (12)
            Type Payload: Transform (3)
                Next payload: Transform (3)
                0... .... = Critical Bit: Not Critical
                Payload length: 8
                Transform Type: Integrity Algorithm (INTEG) (3)
                Transform ID (INTEG): AUTH_HMAC_SHA1_96 (2)
            Type Payload: Transform (3)
                Next payload: Transform (3)
                0... .... = Critical Bit: Not Critical
                Payload length: 8
                Transform Type: Integrity Algorithm (INTEG) (3)
                Transform ID (INTEG): AUTH_HMAC_MD5_96 (1)
            Type Payload: Transform (3)
                Next payload: Transform (3)
                0... .... = Critical Bit: Not Critical
                Payload length: 8
                Transform Type: Diffie-Hellman Group (D-H) (4)
                Transform ID (D-H): 1536 bit MODP group (5)
Type Payload: Transform (3)
                Next payload: NONE / No Next Payload  (0)
                0... .... = Critical Bit: Not Critical
                Payload length: 8
                Transform Type: Diffie-Hellman Group (D-H) (4)
                Transform ID (D-H): 1024 bit MODP group (2)
```

The following illustrates the default IKEv2 proposal from an Cisco IOS IKEv2 initiator, which shows the parameters that are proposed. (This output was obtained by the **show crypto ikev2 proposal** command.)

Click here to view code image

```
IKEv2 proposal: default
     Encryption : AES-CBC-256 AES-CBC-192 AES-CBC-128
     Integrity  : SHA512 SHA384 SHA256 SHA96 MD596
     PRF        : SHA512 SHA384 SHA256 SHA1 MD5
     DH Group   : DH_GROUP_1536_MODP/Group 5 DH_GROUP_1024_MODP/Group 2
```

The responder will select the preferred algorithm that it supports. It can be seen that the most cryptographically secure algorithms are proposed by the initiator in order, which will result in establishing the most cryptographically secure IKEv2 SA.

Security Parameter Index (SPI)

The Security Parameter Index (SPI) is a 64-bit identifier that uniquely identifies the IKEv2 session; this value is created by both parties involved in the IKEv2 session and included in the IKEv2 header.

Because the SPI is unique, the same value must not be used for multiple connections between hosts, since the SPI is used to match incoming IKEv2 sessions and not any other attribute such as IP address, protocol, or port. Because this value is used to identify the IKEv2 session, it is sent in the IKEv2 header in the clear, even when the session has been encrypted (in the IKE_AUTH and subsequent exchanges). Although this value is sent in the clear, the SPI itself is integrity protected and cannot be tampered with by an intruder.

The value of all zeros is only used by the initiator when setting the responder's SPI in the initial IKE_SA_INIT exchange, for this value will not be known by the initiator. The responder will set the SPI to zero if an issue with the initiator's initial connection attempt resulted in the responder sending an INFORMATIONAL, INVALID_KE_PAYLOAD, or NO_PROPOSAL_CHOSEN payload in return. If the responder is implementing the anti-DoS cookie mechanism, the responder's SPI will be set to all zeros.

The following packet output displays the SPI from the initiator in an IKE_SA_INIT exchange; note that the responder's SPI is set to all zeros as the initiator does not yet know the value of this index.

Click here to view code image

```
Internet Security Association and Key Management Protocol
    Initiator SPI: 7803611e4b2ff28c
    Responder SPI: 0000000000000000
    Next payload: Security Association (33)
    Version: 2.0
        0010 .... = MjVer: 0x02
        .... 0000 = MnVer: 0x00
    Exchange type: IKE_SA_INIT (34)
    Flags: 0x08 (Initiator, No higher version, Request)
        .... 1... = Initiator: Initiator
        ...0 .... = Version: No higher version
        ..0. .... = Response: Request
```

The following packet output displays the corresponding reply from the responder, which includes both the initiator's SPI and the responder's.

Click here to view code image

```
Internet Security Association and Key Management Protocol
    Initiator SPI: 7803611e4b2ff28c
    Responder SPI: 5d1b50735fe39ffc
    Next payload: Security Association (33)
    Version: 2.0
        0010 .... = MjVer: 0x02
        .... 0000 = MnVer: 0x00
    Exchange type: IKE_SA_INIT (34)
    Flags: 0x20 (Responder, No higher version, Response)
        .... 0... = Initiator: Responder
        ...0 .... = Version: No higher version
        ..1. .... = Response: Response
```

Nonce

A nonce is a randomly generated value that is used as the input to a cryptographic function. Within IKEv2, this value is used as an input to the secret key generation algorithm and is used as an input to the data that is used in the method to authenticate peers. Adding this value into a cryptographic function ensures that the output is not predictable and adds freshness to material generated for the key exchange.

The nonce must be at a minimum 128 bits in size; it can actually be up to a maximum of 2048 bits in size. The size of the nonce must be at least half the size of the output for the PRF hash function that is negotiated.

As the nonce is sent by the initiator before the PRF is decided on, if it is assumed that multiple PRFs are sent from the initiator, then the size of the nonce must be at least half the size of the largest PRF. For example, if both PRF_HMAC_MD5 and PRF_HMAC_SHA_384 were sent from the initiator, then the smallest nonce would need to be 192 bits, which is half the size of the output of PRF_HMAC_SHA-384 (384 bits).

The following packet output displays a nonce payload taken from an initiator's request. The initiator sent a number of PRFs, including PRF_HMAC_SHA2_512, which results in a 512 bits hash. The size of the payload is 36 bytes; however, this size includes 4 bytes for the header, so the nonce data is 32 bytes, or 256 bits, which is half of the number of bits required for the largest PRF being proposed.

The following packet output displays the nonce payload from a IKE_SA_INIT exchange.

Click here to view code image

```
Type Payload: Nonce (40)
        Next payload: Notify (41)
        0... .... = Critical Bit: Not Critical
        Payload length: 36
        Nonce DATA: 559292633baf45cf133f7ceebf963608c2f481bdc715ac67...
```

Cookie Notification

In an initial IKEv2 exchange, assuming that all IKE_SA_INIT components are correctly accepted by the responder, it will respond with its own parameters, including a Diffie-Hellman public value, and then hold state for this session. This state will use memory and CPU resources. The requirement for the responder to hold state allows for an easy DoS attack by which an attacker sends many IKE_SA_INIT messages to a VPN headend, in which each IKE_SA_INIT request has a spoofed source IP address. On receipt of these spoofed IKE_SA_INIT requests, the headend will need to respond to all requests and hold state for each request until it times out. This attack is very much like a TCP SYN flood, whereby system functions are starved of resources.

This attack is emphasized when an implementation supports RFC6989 and performs cryptographic checks against the received Diffie-Hellman public value to guarantee that attacks cannot be performed to compromise shared key material. These checks are very computationally expensive, and forcing the VPN headend to perform a large number of these can cause a DoS condition.

To mitigate this DoS attack, IKEv2 has a mechanism to prevent blind spoofing attacks. When a certain amount of half connections are seen, the responder can be configured to reply to any IKE_SA_INIT requests with just a cookie notify payload. The initiator will hold no state about the session after sending this cookie, but the cookie will contain information about the state of the request sent from the

initiator.

The cookie is constructed as follows:

Click here to view code image

```
Cookie = <VersionIDofSecret> | Hash (Ni | IPi | SPIi | <secret>)
```

Here *secret* is a random value that is stored locally and changed regularly. Because this value is used in the construction of the cookie and could be used by an attacker to exploit the anti-DoS mechanism, it is a closely guarded secret. The value *VersionIDofSecret* is changed whenever the *secret* is changed. When the initiator receives the cookie notify payload, it must then reinitiate the IKE_SA_INIT exchange to the responder, attaching the cookie notify payload received. On receipt of this payload, the responder can verify that the cookie is legitimate by reconstructing the cookie using the nonce, IP address, and SPI received from the initiator and then substituting the *VersionIDofSecret* and *secret* for the values held locally. If these match, then it can be assumed that the initiator sent a IKE_SA_INIT response that resulted in this cookie being generated.

If an attacker has sent many IKE_SA_INIT requests with spoofed source IP addresses, the reply from the responder would be returned to the spoofed source IP address, which would be discarded on receipt because the spoofed host did not send this request. This behavior, therefore, makes the attack useless.

By default Cisco IOS does not have the cookie notification mechanism enabled. This mechanism can be enabled by the **crypto ikev2 cookie-challenge** *number* command.

Note that cookie notifications only prevent blind spoofing attacks and do not prevent DoS attacks conducted from hosts that use their real IP addresses. If a host initiates an IKE_SA_INIT request that results in the responder generating a cookie notification, then the initiator can simply respond to this (as per the RFC) and so the responder will then perform the expensive cryptographic processing. For this reason care should be taken when designing a VPN solution that allows connections from any IP source address.

Figure 2-4 shows cookie notification in action; the IKE_SA_INIT exchange is increased from two packets to four.

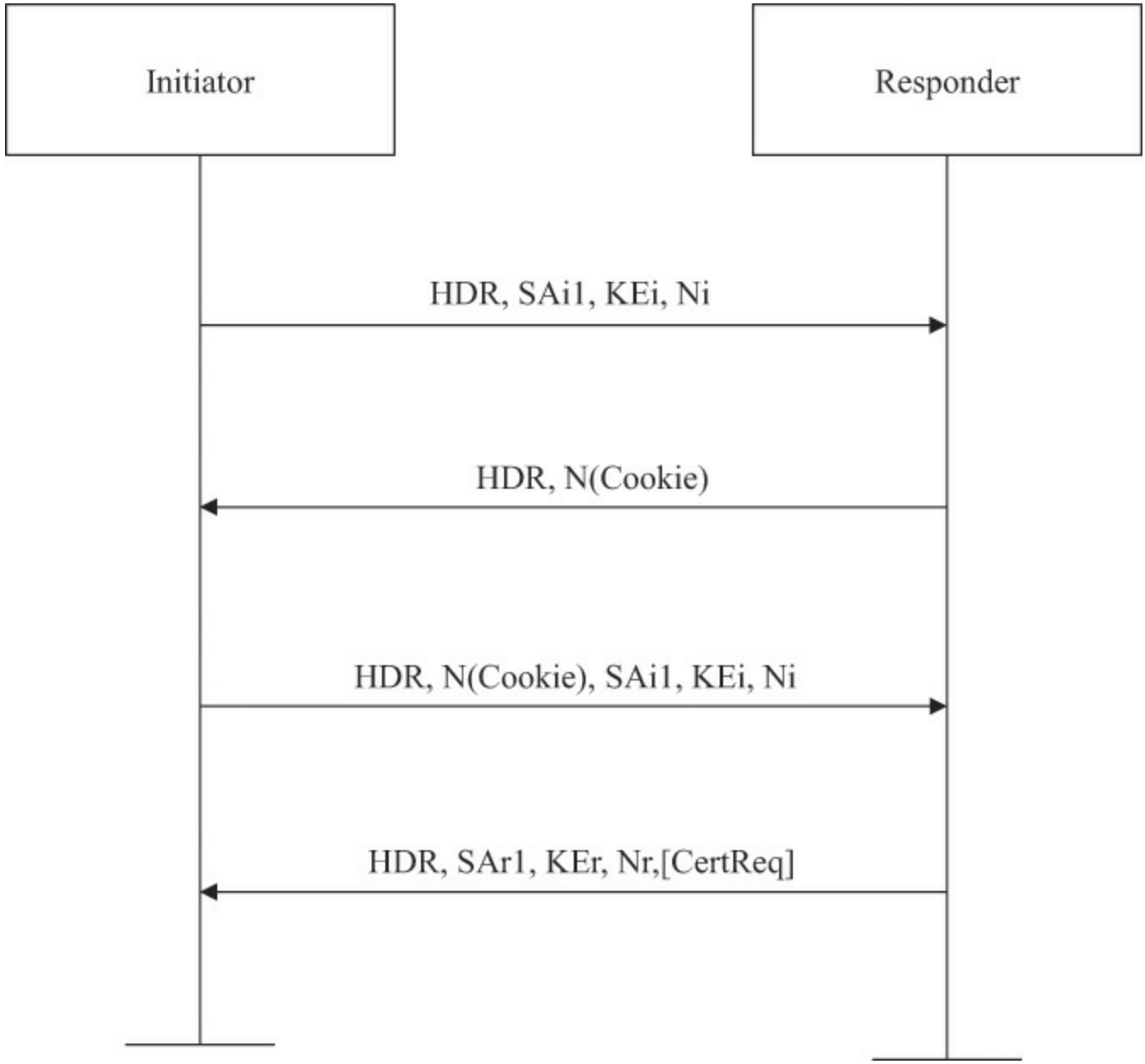


Figure 2-4 *IKE_SA_INIT with Cookie Notification*

Should the responder reject the Diffie-Hellman group proposed by the initiator and cookie notification is activated, then the exchange will increase from four to six packets, Figure 2-5 shows this occurring. The responder will reply to the initiator with a cookie notification, the initiator will then resend the IKE_SA_INIT request with the cookie included, and the responder will then reject the Diffie-Hellman group and propose a new group. The initiator will then send the IKE_SA_INIT request again with the correct Diffie-Hellman group and the cookie.

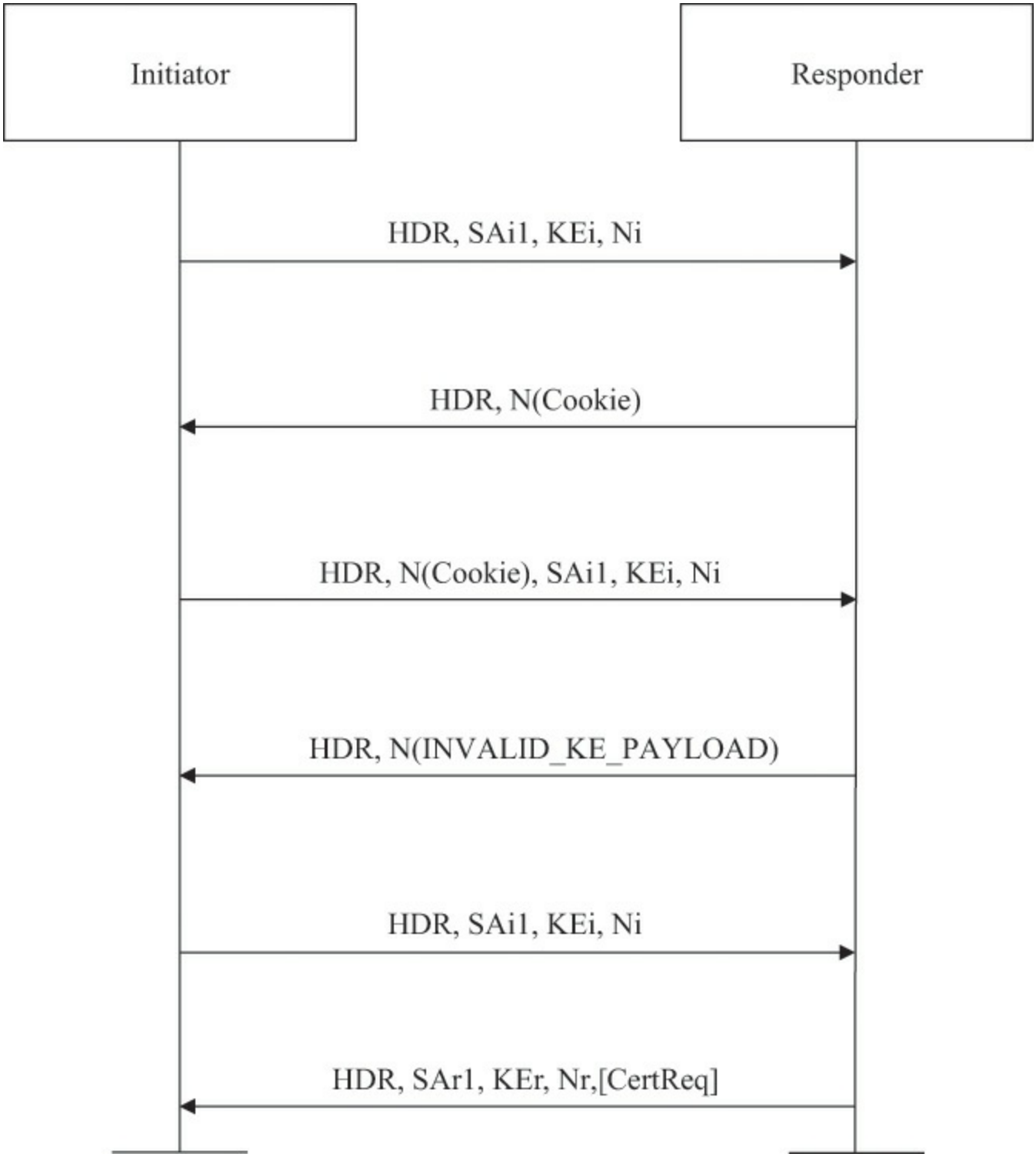


Figure 2-5 *IKE_SA_INIT with Cookie Notification and Invalid Diffie-Hellman Group*

Certificate Request

If a responder is using certificate-based authentication to authenticate peers, then the responder can request that the initiator uses a certificate issued from a preferred certificate authority (CA). This is achieved by sending an SHA-1 hash of the public key of any trusted CA in the IKE_SA_INIT reply message. If multiple trusted CAs are configured on the device, then the hashes of each public key are concatenated into a stream; the peer that receives the certificate request can then select a CA that it supports based on this received value.

This behavior is an improvement over that of IKEv1. In IKEv1 the subject name of the CA was included, which could be duplicated. Having an SHA-1 hash removed this limitation.

The following packet output displays the certificate request payload. There are five hashes sent, so this device will trust five CAs.

Click here to view code image

```
Type Payload: Certificate Request (38)
        Next payload: Notify (41)
        0... .... = Critical Bit: Not Critical
        Payload length: 105
        Certificate Type: Hash and URL of X.509 certificate (12)
        Certificate Authority Data: e27f7bd877d5df9e0a3f9eb4cb0e2ea9efdb6977
        Certificate Authority Data: 3c76492ca10b9f850e2fd6fbae0d362daa2b9c01
        Certificate Authority Data: d0c52226ab4f4660ecae0591c7dc5ad1b047f76c
        Certificate Authority Data: 27f3c8151e6e9a020916ad2ba089605fda7b2faa
        Certificate Authority Data: a6031d7fcabdb29140c6cb82361f6b988fddbc29
```

HTTP_CERT_LOOKUP_SUPPORTED

In any certificate request payload, the sender can notify the peer that it supports the ability to retrieve a certificate via HTTP instead of having the peer send it directly.

This allows the peer to send a 160-bit SHA-1 hash of the certificate along with a URL indicating where the certificate can be obtained. This allows the receiving device to obtain the certificate out of band and does not require potential fragmentation of IKEv2 packets due to large certificates.

The following packet output displays the HTTP_CERT_LOOKUP_SUPPORTED payload. This allows the peer to the send its certificate via a URL.

Click here to view code image

```
Type Payload: Notify (41)
        Next payload: NONE / No Next Payload  (0)
        0... .... = Critical Bit: Not Critical
        Payload length: 8
        Protocol ID: IKE (1)
        SPI Size: 0
        Notify Message Type: HTTP_CERT_LOOKUP_SUPPORTED (16392)
```

This method of authentication is not widely used and is not supported by all software clients.

Key Material Generation

After the IKE_SA_INIT exchange completes, each peer would have agreed on a set of cryptographic algorithms to use and derived a shared secret from the Diffie-Hellman exchange. Once this has occurred, all IKEv2 packets can then be encrypted and integrity protected using the cryptographic algorithms negotiated.

Four cryptographic algorithms are negotiated in the Security Association proposals: an encryption algorithm, an integrity protection algorithm, a Diffie-Hellman group, and a pseudo-random function (PRF).

The two parties must have agreed on the algorithms used, else each party would not be able to generate the key material used to encrypt and decrypt secure communications with the other peer. The initiator would have sent the proposed algorithms, and the responder would have selected the most preferred algorithm that it supports.

A value known as the SKEYSEED is calculated from the nonces exchanged during the IKE_SA_INIT exchange and the Diffie-Hellman shared secret established during that exchange; the calculation is;

SKEYSEED = prf(*Ni* | *Nr*, *g^ir*)

(The | symbol denotes concatenation). *Ni* and *Nr* are the nonce from the initiator and responder respectively, and *g^ir* is the shared secret from the ephemeral Diffie-Hellman exchange.

The PRF is a pseudo-random function that was agreed upon during the IKEv2 exchange; on Cisco IOS MD5 (PRF_HMAC_MD5), SHA-1 (PRF_HMAC_SHA1), SHA256 (PRF_HMAC_SHA2_256), SHA384 (PRF_HMAC_SHA2_2384), or SHA512 (PRF_HMAC_SHA2_512) PRF can be configured. MD5 and SHA-1 are described in RFC2104, and the SHA-2 variants are covered in RFC4868. The PRF is actually a keyed-hash message authentication code (HMAC). This involves a cryptographic hash function combined with a secret cryptographic key (known as the authentication key and defined in RFC2104). This should not be confused with the HMAC used to protect the integrity of IKE and IPsec messages.

RFC2104 and RFC4868 define the use of the HMAC in IKEv2. With H as the cryptographic hash function, *K* the authentication key, and B the block size, *ipad* and *opad are* defined as:

ipad = the byte 0x36 repeated B times

opad = the byte 0x5C repeated B times

To compute an HMAC over the data text we perform

H(*K* XOR *opad*, H(*K* XOR *ipad, text*))

When the SKEYSEED is calculated, the authentication key is generated by concatenating the nonces, Ni and Nr. Although this key would be known to an attacker (as the nonces are exchanged in the IKE_SA_INIT exchange in the clear), the data that is used as the text input (g^ir) would not be known, because it is calculated from the Diffie-Hellman exchange and can be assumed to be cryptographically secure; therefore this calculation can be considered secure.

The PRF used for the SKEYSEED generation will produce a fixed-length output, for this is required of the authentication key to be used for the next PRF. This can be seen by the omission of the "+" symbol when this is described in RFC7296.

The PRF will take a variable-length input and produce a fixed-length output; the output size will vary, depending on the algorithm used.

Table 2-3 defines the PRF algorithms, the block size used, and the output generated.

Algorithm	Block Size (bits)	Output (bits)
PRF_HMAC_MD5	512	128
PRF_HMAC_SHA1	512	160
PRF-HMAC-SHA-256	512	256
PRF-HMAC-SHA-384	1024	384
PRF-HMAC-SHA-512	1024	512

Table 2-3 *PRF Algorithms*

Once the SKEYSEED has been generated, it is then used as the authentication key input into another PRF that is then used to derive the key material for the IKEv2 SA. The calculation is:

prf+ (SKEYSEED, Ni | Nr | SPIi | SPIr)

This function is used to generate a string of key material that will be used for various cryptographic functions that protect the IKEv2 SA. Because the PRF outputs a fixed-length value, if the length of the key material required is longer than the output of the PRF, multiple PRFs must be used. The inclusion of the "+" symbol to the PRF denotes that a variable-length output is required.

The calculation is shown below:

prf+ (*K*,*S*) = T1 | T2 | T3 | T4 | ...

where:

T1 = prf (*K*, *S* | 0x01)

T2 = prf (*K*, *T1* | *S* | 0×02)

T3 = prf (*K*, *T2* | *S* | 0×03)

T4 = prf (*K*, *T3* | *S* | 0×04)

Note that *K* = *SKEYSEED* and *S* = *Ni* | *Nr* | *SPIi* | *SPIr*

The key material generated is a string that can be split to gain the following values

{*SK_d* | *SK_ai* | *SK_ar* | *SK_ei* | *SK_er* | *SK_pi* | *SK_pr* }

- *SK_d* is used for deriving new keys for the Child SAs.
- *SK_ai* and *SK_ar* are used as a key to the integrity protection algorithm for authenticating the component messages of subsequent exchanges.
- *SK_ei* and *SK_er* are used for encrypting (and of course decrypting) all subsequent IKEv2 payloads.
- *SK_pi* and *SK_pr*, are used as the key input when authenticating an AUTH payload. The lengths of *SK_d, SK_pi* and *SK_pr* must be the preferred key length of the PRF agreed upon.

Once the key material has been generated (after the IKE_SA_INIT), all IKEv2 exchanges (IKE_AUTH, INFORMATIONAL, CREATE_CHILD_SA, etc.) are protected by the cryptographic mechanisms negotiated and key material generated.

IKE_AUTH

Once the IKE_SA_INIT exchange has completed successfully, the IKE_AUTH exchange will occur. This exchange is encrypted and authenticated, and allows both parties to authenticate themselves. Additionally, a single pair of IPsec Security Associations is created.

Encrypted and Authenticated Payload

The IKE_AUTH exchange is required to be encrypted and authenticated. The first IKEv2 payload within this exchange is the Encrypted/Authenticated payload type, which denotes that the exchange is protected.

This payload will include an initialization vector (IV) that is used as the input to the encryption function. This IV is sent in the clear and must be unique for each payload. The encrypted data will contain the remaining IKEv2 payloads in encrypted format. There is a possibility that padding might be required to match the encryption block size; this padding will be added if required. If a combined mode cipher is not used for encryption, an integrity checksum is added at the end of the header; the length of this checksum corresponds to the integrity algorithm used.

The following example displays an encrypted and authenticated payload. The initialization vector can be seen, along with the encrypted data. At the end of the payload is the integrity checksum, which protects the whole of the IKE message, including the clear text headers and the encrypted data, from being modified.

Click here to view code image

```
Type Payload: Encrypted and Authenticated (46)
    Next payload: Security Association (33)
    0... .... = Critical Bit: Not Critical
    Payload length: 160
    Initialization Vector: 36a1dd71fbeaaa683c2254e8530aa781 (16 bytes)
    Encrypted Data (128 bytes)
        Padding (11 bytes)
        Pad Length: 11
    Integrity Checksum Data: ef3e7c44eff24d09902ebcfc (12 bytes)[correct]
```

Figure 2-6 shows the IKE_AUTH exchange.

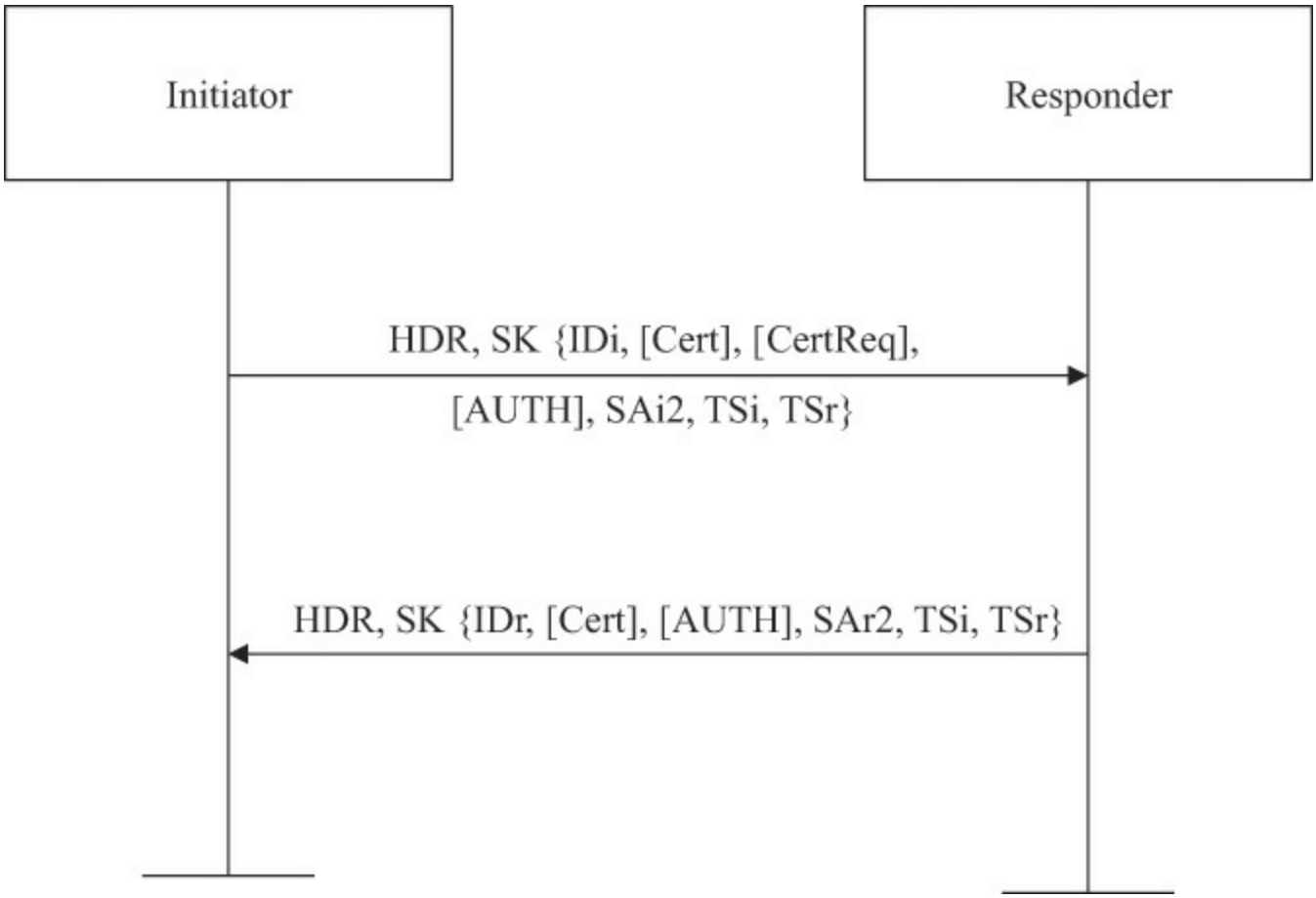


Figure 2-6 *IKE_AUTH Exchange*

The following describes each of the parameters of the IKE_AUTH exchange.

- **SK { ... }** : Payload between { } is encrypted and integrity protected
- **IDi** : Initiator identity
- **IDr** : Responder identity
- **Cert** : Initiator certificate (optional)
- **CertReq**: Certificate request (optional)
- **AUTH**: Initiator and responder authentication data (not present if initiator authenticates via EAP)

- **SAi2**: Child SA transforms proposed by initiator
- **SAr2**: Child SA transforms chosen by responder
- **TSi/TSr**: Child SA traffic selectors (src/dst proxies)

Encrypted Payload Structure

After the IKE_SA_INIT exchange, all messages will be encrypted, and integrity will be protected. The IKE header is integrity protected, although all information in it is sent in the clear. However, all data within the next payload, known as the Encrypted and Authenticated payload, will be encrypted and integrity protected using the algorithms negotiated and keys derived in the IKE_SA_INIT exchange.

The following shows an IKE_AUTH message. The headers can be seen with the IKE version, SPI, and flags; however the payload is encrypted, so it cannot be seen by someone who lacks the keys to decrypt it.

Click here to view code image

```
User Datagram Protocol, Src Port: 500 (500), Dst Port: 500 (500)
Internet Security Association and Key Management Protocol
    Initiator SPI: a3ae32c968f6ccf0
    Responder SPI: fc1c21ef24ef360c
    Next payload: Encrypted and Authenticated (46)
    Version: 2.0
    Exchange type: IKE_AUTH (35)
    Flags: 0x08 (Initiator, No higher version, Request)
    Message ID: 0x00000001
    Length: 572
    Type Payload: Encrypted and Authenticated (46)
```

Identity

Once the IKEv2 session is secure, both parties can reveal their identity. The IKEv2 protocol allows for both parties' identities to be protected from view because these are sent within the encrypted IKE_AUTH exchange. The identity is sent within a specific identity payload that denotes the type of identity, along with the identity itself.

This identity will be validated by the peer device when authentication is performed. When preshared keys are used, each peer should have a database of shared keys that relate to an identity. When a device sends its identity along with authentication data, the receiving peer can validate that the peer is who it claims to be by extracting the relevant shared key from the database. When digital signatures are used as a means of authentication, the distinguished name can be extracted from the certificate and sent as the identity in Distinguished Encoding Rules (DER) format. This identity can be checked against the presented certificate and validated when the cryptographic signature checks are performed.

RFC7296 does not actually stipulate that a device that uses a digital signature algorithm must use an identity taken from the certificate. The identity can be anything: for example, an IP Address, FQDN, or even email address. Likewise, if pre-shared keys are used, an identity of FQDN or email address can be used, or the identity type of IPv4 or IPv6 could be used. However, it does not have to be.

Table 2-4 lists the identities supported by RFC7296.

Identity	Description
ID_IPV4_ADDR	A single four (4) octet IPv4 address.
IP_FQDN	A fully-qualified domain name string. An example of an ID_FQDN is "example.com"
ID_RFC822_ADDR	A fully-qualified RFC822 email address string. An example of a ID_RFC822_ADDR is "jsmith@example.com."
ID_IPV6_ADDR	A single sixteen (16) octet IPv6 address.
ID_DER_ASN1_DN	The binary Distinguished Encoding Rules (DER) encoding of an ASN.1 X.500 Distinguished Name
ID_DER_ASN1_GN	The binary DER encoding of an ASN.1 X.509 GeneralName
ID_KEY_ID	An opaque octet stream that may be used to pass vendor-specific information necessary to do certain proprietary types of identification.

Table 2-4 *IKEv2 Identities*

Unless the authentication method in use is EAP, the initiator will always reveal its identity before the responder.

Each party must be configured to accept connections from a peer presenting the specified identity.

Although the identity is protected within the IKE_AUTH exchange, it is worth nothing that an inline intruder could reply to an initiator's IKE_SA_INIT request by spoofing the responder's IP address. As long as the attacker supported the cryptographic algorithms and Diffie-Hellman parameters proposed, the attacker would be able to successfully decode the received IKE_AUTH and so view the initiators identity.

Authentication

Once the key material has been generated to secure the IKEv2 session, each device must authenticate the peer device. After the IKE_SA_INIT exchange has taken place, each device will then have a secure channel (the IKE_AUTH exchanges being secured) and the necessary attributes to validate itself with the peer.

Three methods of authentication are used in IKEv2: signature, pre-shared key, and EAP. Each peer within an exchange can use a local authentication method, so the initiator could use pre-shared key, and the responder could use signature.

It is a rule that if EAP is used, the responder must use signature-based authentication.

Authentication data is included within an Authentication payload. This payload contains the authentication method used along with the authentication data that is used to validate the device sending this data.

The Diffie-Hellman exchange is vulnerable to a man-in-the-middle attack, where an attacker can intercept messages from both parties and reply with its own values. Therefore the authentication exchange includes attributes that uniquely identify the hosts in the session and the attributes relating to this such as the IKE Identity, nonce, and SPI.

Signature-Based Authentication

Signature-based authentication provides authenticity by requiring a client to sign a block of data that is constructed from key material and data used in the previous (IKE_SA_INIT) and current (IKE_AUTH) exchanges; this combined information is referred to as signed octets.

Should the initiator use signature-based authentication, then the signed octets can be described as:

Click here to view code image

```
InitiatorSignedOctets = RealMessage1 | NonceRData | MACedIDForI    GenIKEHDR = [
four octets 0 if using port 4500 ] | RealIKEHDR
RealIKEHDR =  SPIi | SPIr |  . . . | Length
RealMessage1 = RealIKEHDR | RestOfMessage1
NonceRPayload = PayloadHeader | NonceRData
InitiatorIDPayload = PayloadHeader | RestOfInitIDPayload
RestOfInitIDPayload = IDType | RESERVED | InitIDData
MACedIDForI = prf(SK_pi, RestOfInitIDPayload)
```

Note that SK_pi is generated earlier from the SKEYSEED material.

The initiator signs the first message (the IKE_SA_INIT request), starting with the first octet of the first SPI in the header and ending with the last octet of the last payload. Appended to this (for purposes of computing the signature) are the responder's nonce Nr, and the value prf(SK_pi, IDi). It is critical to the security of the exchange that each side sign the other side's nonce to add randomness to the exchange.

When the responder receives these signed octets, it can decrypt these using the related public key that is contained within the certificate presented by the initiator in the certificate payload and can then verify that the holder of the presented certificate is in fact the true owner.

Should the responder use signature-based authentication, then the signed octets can be described as:

Click here to view code image

```
ResponderSignedOctets = RealMessage2 | NonceIData | MACedIDForR    GenIKEHDR = [
four octets 0 if using port 4500 ] | RealIKEHDR
RealIKEHDR =  SPIi | SPIr |  . . . | Length
RealMessage2 = RealIKEHDR | RestOfMessage2
NonceIPayload = PayloadHeader | NonceIData
ResponderIDPayload = PayloadHeader | RestOfRespIDPayload    RestOfRespIDPayload =
IDType | RESERVED | RespIDData
MACedIDForR = prf(SK_pr, RestOfRespIDPayload)
```

When the initiator receives these signed octets, it can decrypt these using the related public key that is contained within the certificate presented by the responder and verify that the holder of the presented certificate is in fact the true owner.

(Pre) Shared-Key-Based Authentication

The use of a shared-key (also known as Pre-Shared Key) between hosts is simple to setup, but is known to be cryptographically weaker than using certificates or EAP. If a shared key is used, the key should be cryptographically strong, containing enough entropy to counter a dictionary-based attack. This is described within RFC7296.

Note

It is a common but typically insecure practice to have a shared key derived solely from a user-chosen password without incorporating another source of randomness. This is typically insecure because user-chosen passwords are unlikely to have sufficient unpredictability to resist dictionary attacks, and these attacks are not prevented in this authentication method.

The method for a peer to prove authenticity is for it to provide the output of a pseudorandom function (negotiated from the IKE_SA_INIT exchange) containing attributes from the IKE_SA_INIT and IKE_AUTH exchanges plus the shared secret.

For the initiator:

Click here to view code image

```
AUTH = prf( prf(Shared Secret, "Key Pad for IKEv2"), <InitiatorSignedOctets>)
```

where the InitiatorSignedOctets are the same as used in signature-based authentication.

The initiator will send the value generated for AUTH as the authentication data in the authentication payload.

For the responder:

Click here to view code image

```
AUTH = prf( prf(Shared Secret, "Key Pad for IKEv2"), <ResponderSignedOctets>)
```

where ResponderSignedOctets are the same as used in signature-based authentication.

The initiator will send the value generated for AUTH as the authentication data in the authentication payload.

It is worth noting that when a shared secret is used, if the initiator is tricked into connecting to a rogue VPN headend or to a MITM masquerading as the true responder, when the initiator sends the authentication data an attacker will know all attributes apart from the shared secret. An attacker would be able to capture the AUTH payload and perform an offline brute-force attack, entering the correct (known) data for <ResponderSignedOctets> and the string “Key Pad for IKEv2” while substituting values for the shared secret (for example, using a dictionary attack) until the same value matched. If the shared secret was a weak value that used a dictionary word or a string with little entropy, this would be easily breakable with current computing power.

As IKEv2 can allow asymmetric authentication, it should be noted that one peer could use certificate-based authentication, while the other peer uses shared key, or both parties could use separate shared secrets.

EAP

Extensible Authentication Protocol (EAP) (RFC3748) is a valid method of authenticating an initiator within IKEv2. Should a host use EAP as the authentication method, then the peer must use a method of authentication based on a public key signature. For this reason if EAP is required in Cisco IOS, one peer must support an RSA or ECDSA method of authentication. If EAP is to be used, it is common for the responder to authenticate the initiator using EAP, which relates well to a remote access VPN deployment where the headend device can authenticate remote clients using EAP via a backend RADIUS database.

When a host uses shared-key or signature-based authentication, the Authentication payload contains data to authenticate that validity of the host. As EAP is a separate protocol to IKEv2, should a host wish to authenticate using EAP, then it will exclude the Authentication payload from the IKE_AUTH exchange. RFC7296 states that the initiating host includes an identification payload, which contains the identity that the host is claiming to be. This identity will be proved when using EAP or just used for AAA purposes, EAP will actually use a second identity.

Figure 2-7 shows the IKE_AUTH exchange when using EAP where the IKE identity is used as the EAP identity.

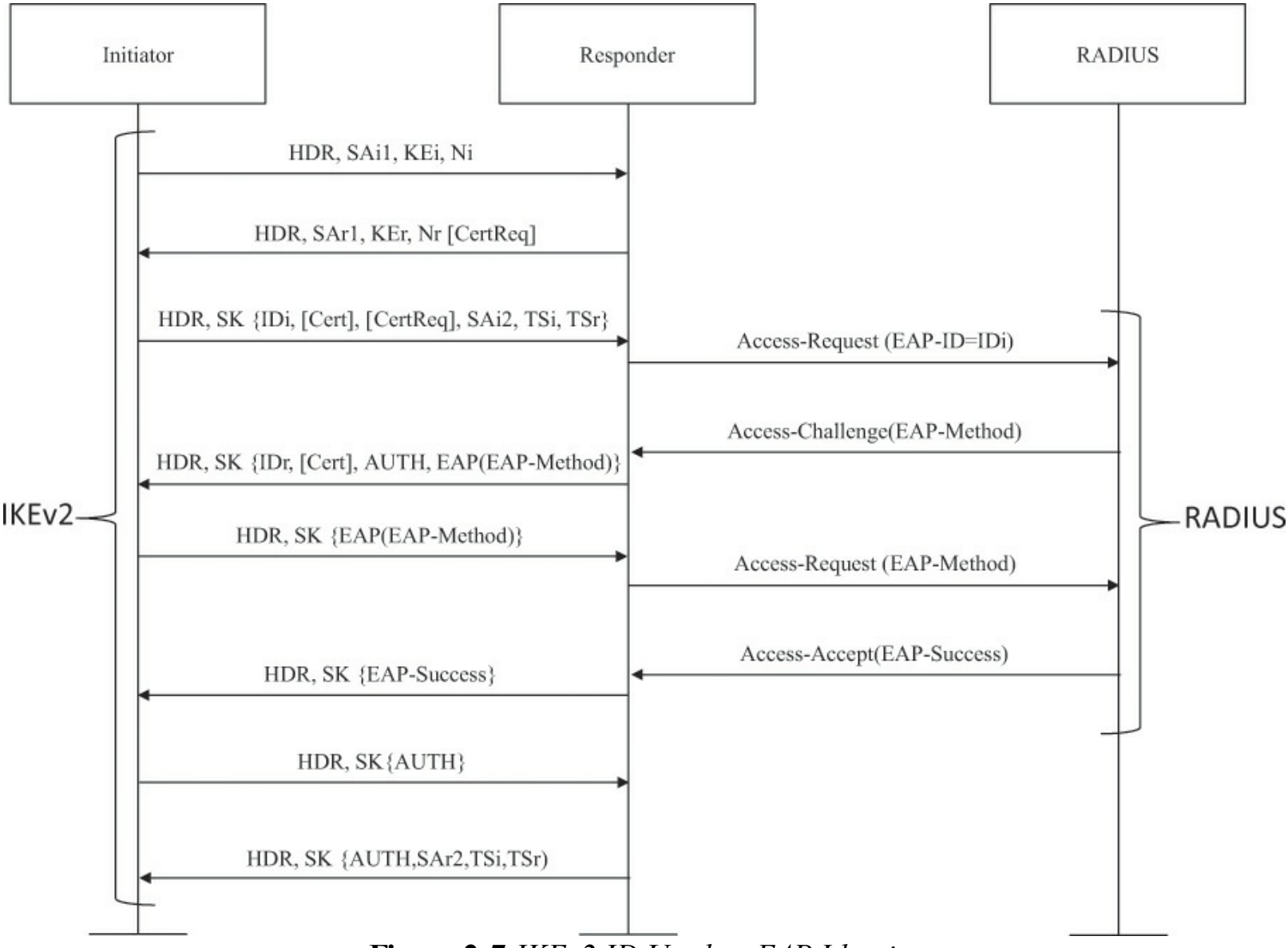


Figure 2-7 *IKEv2 ID Used as EAP Identity*

Cisco IOS allows for the EAP identity to be queried by the responder by using the **query-identity**

command. In this case the identity presented by the initiator will not be used, but instead the identity will be queried within the EAP exchange. This overcomes limitations in cases when clients are required to request an EAP Identity request before any other EAP requests. Microsoft Windows 7 is an example of such a client.

> **Note**
>
> that an EAP identity cannot be an IP address, so if the initiator's identity is an IPv4 or IPv6 address and the **query-identity** command is not used on Cisco IOS, then the exchange will fail.

Figure 2-8 shows the IKE_AUTH exchange when EAP is used with the **query-identity** command configured. Rather than using the IKE Identity that is received from the initiator, the responder will reply to the IKE_AUTH with a EAP-Request for the EAP identity; this allows for separate IKE and EAP identities to exist for a device.

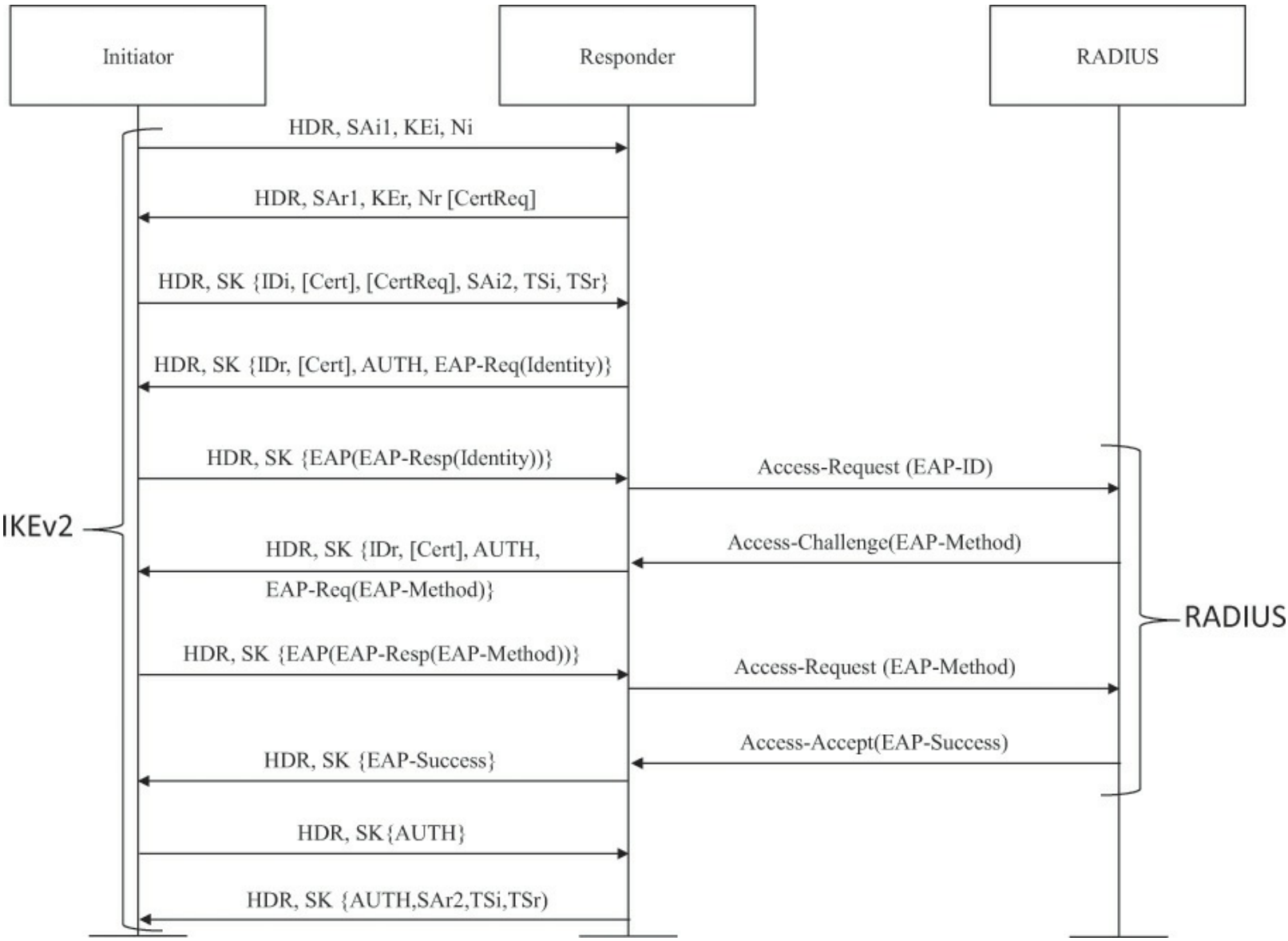


Figure 2-8 *EAP with Query-Identity Used*

If the responder is willing to use an EAP method, it will include an EAP payload (containing an EAP-Request) in the response of the IKE_AUTH exchange and defer sending the SA and traffic selectors until initiator authentication is complete. These will be sent in a subsequent IKE_AUTH exchange.

It is interesting to note that when EAP is used, although the initiator has revealed its identity but not proven this using the EAP exchange, the responder will reveal its identity.

If we assume a minimal EAP exchange, the next exchange from the initiator will contain the EAP-Response. Assuming that the initiator provided valid authenticity then the responder will send an EAP Success message. The initiator will then need to include the AUTH payload containing material to authenticate itself. The method used to generate the Authentication data is the same as that used to create a shared secret; however, because there are two types of EAP authentication methods (key-generating and non-key-generating), the method for authentication changes slightly. For key-generating EAP method, the shared secret is substituted with the master session key that was generated from the EAP exchange. For non- key-generating EAP method, SK_pi and SK_pr are replaced with the shared secret in the authentication material generation.

Traffic Selectors

Within the IKE_AUTH exchange, the peers will exchange what traffic is to be encrypted by the IPsec Security Association. This is achieved by sending one or more traffic selectors that define the encrypted traffic flows between peers.

The initiator will send the traffic selectors containing traffic that it will look to protect and also traffic selectors of traffic it expects to receive from the peer device. Assuming that the responder accepts these traffic selectors, it would respond with a mirror image of the proposed traffic selectors that it received.

Each traffic selector consists of a traffic selector type that denotes either IPv4 or IPv6 address, a range of IP addresses (IPv4 or IPv6), a protocol identifier or ID, and a range of ports that will be used to create the unidirectional IPsec Security Association.

The range of IP addresses consists of two addresses, a beginning and an ending address; any other addresses that fall within this range will be included within the SA.

The protocol ID defines the IP protocol number that represents the IP protocol that is being transported, for example, 6 for TCP or 17 for UDP. 0 denotes that all protocols shall be encrypted.

The full list of IP protocol numbers can be found at;

http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml

The range of ports consists of a start and end port number; any ports in between will be included. As ICMP does not, strictly speaking, use port numbers but type and code values, these are treated as a single 16-bit integer port number, with type in the most significant eight bits and code in the least significant eight bits.

The following example displays the traffic selectors from an initiator. In this example, all IPv4 traffic from the 192.168.2.0/24 subnet will be protected going to the peer subnet 192.168.1.0/24. The protocol ID is set to zero, which denotes all protocols, and the port range is set to all ports (any value between 0 and 65535). This configuration on Cisco IOS would only be possible using a crypto map approach.

Click here to view code image

```
Type Payload: Traffic Selector - Initiator (44) # 1
                    Next payload: Traffic Selector - Responder (45)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 24
```

```
                    Number of Traffic Selector: 1
                    Traffic Selector Type: TS_IPV4_ADDR_RANGE (7)
                    Protocol ID: Unused
                    Selector Length: 16
                    Start Port: 0
                    End Port: 65535
                    Starting Addr: 192.168.2.0 (192.168.2.0)
                    Ending Addr: 192.168.2.255 (192.168.2.255)

Type Payload: Traffic Selector - Responder (45) # 1
                    Next payload: NONE / No Next Payload  (0)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 24
                    Number of Traffic Selector: 1
                    Traffic Selector Type: TS_IPV4_ADDR_RANGE (7)
                    Protocol ID: Unused
                    Selector Length: 16
                    Start Port: 0
                    End Port: 65535
                    Starting Addr: 192.168.1.0 (192.168.1.0)
                    Ending Addr: 192.168.1.255 (192.168.1.255)
```

IKEv2 actually allows the initiator to send a set of traffic selectors and the responder to select a sub-set of these to use. This is not a common case; however, it should be noted.

Note that Cisco IOS does not allow for multiple traffic selectors to exist under an IPsec Security Association. So for every pair of traffic selectors, a separate IPsec Security Association is needed.

When GRE is used to encapsulate IPsec traffic, the GRE traffic will be sourced from the GRE local transport address (also known as the tunnel source address), encapsulated in IPsec, and sent to the tunnel destination. As the traffic selector defines what traffic is to be encrypted, this will contain GRE traffic from the tunnel source to the tunnel destination. The following example displays the traffic selectors used in an IPsec GRE session between two IPv6 peers, 2003::3 and 2007::14.

Click here to view code image

```
Type Payload: Traffic Selector - Initiator (44) # 1
                    Next payload: Traffic Selector - Responder (45)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 24
                    Number of Traffic Selector: 1
                    Traffic Selector Type: TS_IPV6_ADDR_RANGE (8)
                    Protocol ID: 47
                    Selector Length: 40
                    Start Port: 0
                    End Port: 65535
                    Starting Addr: 2003::3 (2003::3)
                    Ending Addr: 2003::3 (2003::3)

Type Payload: Traffic Selector - Responder (45) # 1
                    Next payload: NONE / No Next Payload  (0)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 24
                    Number of Traffic Selector: 1
                    Traffic Selector Type: TS_IPV6_ADDR_RANGE (8)
                    Protocol ID: 47
                    Selector Length: 40
                    Start Port: 0
                    End Port: 65535
                    Starting Addr: 2007::14 (2007::14)
```

```
Ending Addr: 2007::14 (2007::14)
```

Initial Contact

Initial contact allows a host to create an IKEv2 session with a peer and inform that peer that any previous sessions are now invalid and should be removed.

This is the prefect remedy if a host crashes and restarts or if it is manually reset for some reason, or in any case in which the host will lose all information about the previous IKEv2 SA. If such cases, there is no point in the peer sending encrypted IKEv2 and Encapsulation Security Payload/Authentication Header traffic to the host that has restarted, because this data will not be correctly decrypted (because no session keys are held) and so traffic will be discarded, wasting network bandwidth and resource. Initial contact is performed when a host sends an INITIAL_CONTACT notification within the IKE_AUTH exchange; on successful authentication, the peer that received the INITIAL_CONTACT payload will remove any existence of any previous IKEv2 SA state along with any IPsec sessions that relate to that peer, asserting that the latest IKEv2 SA is the only IKEv2 SA currently active between both parties.

The following packet output displays the INITIAL_CONTACT payload, notice that this is contained within a Notify payload.

Click here to view code image

```
Type Payload: Notify (41)
                    Next payload: Identification - Responder (36)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 8
                    Protocol ID: RESERVED (0)
                    SPI Size: 0
                    Notify Message Type: INITIAL_CONTACT (16384)
```

CREATE_CHILD_SA

The CREATE_CHILD_SA exchange is used in three scenarios:

1. For creating additional IPsec Security Associations after the initial IKE exchange, where a CREATE_CHILD_SA exchange is used to create every additional pair of IPsec Security Associations required by the system.
2. When the IPsec Security Association is rekeyed where a new IPsec Security Association is created.
3. If the IKEv2 SA is required to be rekeyed and a new IKEv2 SA is established.

It is worth noting that when new IPsec Security Associations or IKEv2 SAs are created, no authentication material is exchanged. Once the IKEv2 SA is established after it has been successfully authenticated, it will never be authenticated again (unless the IKEv2 SA is cleared and renegotiated using the IKE_SA_INIT/IKE_AUTH exchanges).

The exchange of CREATE_CHILD_SA may be initiated by either party after the initial exchanges are completed.

When a IKEv2 SA or IPsec Security Association is required to be rekeyed, the new SA will (or should) become active before the current SA expires. This allows for a transition from the current SA to the new SA.

IPsec Security Association Creation

Should an IKEv2 session be created that is used to secure more than one IPsec Security Association, the first SA is constructed from the IKE_AUTH exchange, and additional SAs are built from CREATE_CHILD_SA exchanges. The CREATE_CHILD_SA exchange will contain all the information needed to construct additional IPsec Security Associations, including a Security Association payload to exchange cryptographic algorithms, nonces to add randomness, and traffic selectors to denote the traffic that is to be protected. Additionally, a new exchange of Diffie-Hellman public values can be performed, which will be used to construct fresh key material to protect the SA. This is commonly referred to as Forward Secrecy or Perfect Forward Secrecy (PFS).

Keying material for them is generated as follows:

Keymaterial = prf+(*SK_d, Ni | Nr*)

where Ni and Nr are the nonces from the IKE_SA_INIT exchange if this request is the first Child SA created or the fresh Ni and Nr from the CREATE_CHILD_SA exchange if this is a subsequent creation.

For CREATE_CHILD_SA exchanges that include an optional Diffie-Hellman public key exchange, the keying material is defined as:

Keymaterial = prf+(*SK_d*, *g^ir* (new) | *Ni* | *Nr*)

where *g^ir* (new) is the shared secret from the ephemeral Diffie-Hellman exchange.

The "+" appended to the end of the PRF denotes that the output of the PRF can be bit string that is potentially longer than the output of the PRF.

IPsec Security Association Rekey

Before an IPsec Security Association expires (due to the maximum traffic being transferred or lifetime exceeded), the current SA will need to be rekeyed. This actually involves creating a new IPsec Security Association. The CREATE_CHILD_SA exchange is used to negotiate all parameters to construct this new IPsec Security Association. A notify payload of type REKEY_SA is included in the CREATE_CHILD_SA exchange, which denotes that this is used to rekey an IPsec Security Association.

When a rekey occurs, cryptographic algorithms are exchanged using the Security Association payloads. Cisco IOS uses the same algorithms that were used for the parent SA. Nonces are exchanged to add freshness to the generation of key material, and if it is configured, an optional Diffie-Hellman exchange is performed to generate a new shared secret. The traffic that is to be protected by this new SA is exchanged in traffic selector payloads.

When an IPsec Security Association is rekeyed, the same method of generating key material is performed as when a Child SA is created using a CREATE_CHILD_SA exchange. Please refer to the previous description within IPsec Security Association creation for details.

Once the IPsec Security Association has been rekeyed, the previous functioning IPsec Security Association will be deleted by means of an INFORMATIONAL exchange.

IKEv2 Security Association Rekey

An IKEv2 SA is rekeyed using the CREATE_CHILD_SA exchange.

Each peer will exchange the cryptographic algorithms within the Security Association payload, nonces for freshness of the generation of key material, and also new Diffie-Hellman public key values to generate a new shared secret.

A new set of initiator and responder SPIs are supplied in the SPI fields in the proposal structures within the SA payloads. Note that the traffic selector payloads are not required when rekeying an IKEv2 SA, for no IPsec Security Associations are created.

The SKEYSEED for the new IKE SA is computed using SK_d from the existing IKEv2 SA as follows:

SKEYSEED = prf(*SK_d* (old), *g^ir* (new) | *Ni* | *Nr*)

The SKEYSEED can then be used to generate specific key material as needed.

Once both parties have successfully performed the CREATE_CHILD_SA exchange, the new IKEv2 SA can be used.

Any Child SAs that were created by the old IKEv2 SA will be transferred to the new (recently created) IKEv2 SA.

Once the IKEv2 SA has been rekeyed, the old SA will be deleted by either peer sending an INFORMATIONAL exchange.

Note that no authentication occurs when an IKEv2 SA is rekeyed. RFC4739, Multiple Authentication Exchanges in the Internet Key Exchange (IKEv2) Protocol, describes a method for reauthentication in IKEv2; however this is not implemented by Cisco IOS at the time of writing.

IKEv2 Packet Structure Overview

The IKEv2 packet is constructed of an IKEv2 header plus a number of payloads.

The IKEv2 header will always begin with the Security Parameter Indexes; this will allow the receiving host to identify the IKEv2 session as the source IP address can be changed via a NAT device, and also multiple IKEv2 sessions can coexist between the same peers using the same 5 tuple (local and remote IP address, protocol and ports), with each session identified by the Security Parameter Index.

After the IKEv2 header will be a number of IKEv2 payloads, and the payload type is defined by the Next Header field. This allows an IKEv2 packet to contain multiple payloads.

The IKEv2 header will contain a major version number which should be set to 2 (being IKEv2), there is also a minor version number which at the time of writing would be set to 0.

The Exchange Type denotes the type of exchange: IKE_SA_INIT, IKE_AUTH, CREATE_CHILD_SA, or INFORMATIONAL. The data for these exchange types is contained in the latter part of the packet.

Each packet is numbered, which is denoted by the Message ID field. This is always 0 for the IKE_SA_INIT exchange and will increase by 1 for each subsequent exchange. It should be noted that the counter will increase for each exchange, so even if the IKE_SA_INIT exchange contains the Cookie payload and increases the number of packets from two to four, the numbering will still be set to 0 in the Message ID field for all packets sent. The counter is used with IKEv2 windowing; the window size will set a fixed number of messages that can be in flight at any given time.

The counter will be reset to 0 when the IKEv2 SA is rekeyed.

The numbering of packets allows each peer to track the request-response nature of IKEv2 and allows

for retransmissions to easily be identified.

The following UDP output displays an IKE_SA_INIT exchange; this is of course encapsulated in UDP using port 500. The parameters for the IKEv2 header can be seen.

Click here to view code image

```
User Datagram Protocol, Src Port: 500 (500), Dst Port: 500 (500)
    Source Port: 500 (500)
    Destination Port: 500 (500)
    Length: 408
    Checksum: 0xc089
Internet Security Association and Key Management Protocol
    Initiator SPI: 9ff10b12e411fc0f
    Responder SPI: 0000000000000000
    Next payload: Security Association (33)
    Version: 2.0
        0010 .... = MjVer: 0x02
        .... 0000 = MnVer: 0x00
    Exchange type: IKE_SA_INIT (34)
    Flags: 0x08 (Initiator, No higher version, Request)
        .... 1... = Initiator: Initiator
        ...0 .... = Version: No higher version
        ..0. .... = Response: Request
    Message ID: 0x00000000
    Length: 400
```

The INFORMATIONAL Exchange

The INFORMATIONAL exchange is used to communicate control messages between peers. This exchange will contain one or more message type payloads; Notification, Delete, and/or Configuration, with each allowing reporting of errors, notification of events or exchange of configuration information.

There is actually one case where an INFORMATIONAL exchange will not contain any payloads; in this case it is used as a keepalive packet to check that the peer is up and active.

An INFORMATIONAL exchange that relates to an established IKEv2 SA will be cryptographically protected by the algorithms used to protect the SA, so the exchange will be sent after the IKE_AUTH exchange has occurred.

The exception to all IKEv2 messages being in pairs occurs when an INFORMATIONAL message is sent when IPsec traffic is received from a peer and the receiving peer does not have a matching IPsec Security Association. All other INFORMATIONAL exchanges consist of a pair of messages, with the second acknowledging the first.

Notification

The Notification payload, (or Notify) is used to signal status about events pertaining to the IKEv2 session. Examples include error conditions and attempts to negotiate unsupported attributes. A full list can be obtained from:

> http://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml#ikev2-parameters-14

Deleting Security Associations

Once an IKEv2 session has been rekeyed and a new IKEv2 SA created, the original can be removed. The last exchange by the initiator deletes the old IKE SA by sending an INFORMATIONAL exchange that includes the Delete payload.

The INFORMATIONAL exchange containing the Delete payload will also occur should a device have to proactively bring the IKEv2 SA down; for example if an IKEv2 SA is manually cleared on a device or if the corresponding Cisco IOS tunnel interface is shut down, this will trigger the INFORMATIONAL exchange containing a Delete payload to notify the peer that the session has been destroyed.

The following packet output displays an INFORMATIONAL exchange being used to delete an IPsec Security Association. The IPsec SPI can be seen as cb989cce.

Click here to view code image

```
Internet Security Association and Key Management Protocol
    Initiator SPI: 1739c43d75f18743
    Responder SPI: 1f9bde873d9281f5
    Next payload: Encrypted and Authenticated (46)
    Version: 2.0
    Exchange type: INFORMATIONAL (37)
    Flags: 0x08 (Initiator, No higher version, Request)
    Message ID: 0x00000005
    Length: 76
    Type Payload: Encrypted and Authenticated (46)
        Next payload: Delete (42)
        0... .... = Critical Bit: Not Critical
        Payload length: 48
        Initialization Vector: 459cee48eae9966ac0be082687da717f (16 bytes)
        Encrypted Data (16 bytes)
        Decrypted Data (16 bytes)
            Contained Data (12 bytes)
                Type Payload: Delete (42)
                    Next payload: NONE / No Next Payload  (0)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 12
                    Protocol ID: ESP (3)
                    SPI Size: 4
                    Port: 1
                    Delete SPI: cb989cce
            Padding (3 bytes)
            Pad Length: 3
        Integrity Checksum Data: ccb4a8512ff5ef126611b4b3 (12 bytes)[correct]
```

Similar to Encapsulation Security Payload and Authentication Header SAs, IKE SAs are also deleted by sending an informational exchange. Deleting an IKE SA implicitly closes any remaining Child SAs negotiated under it. The response to a request that deletes the IKE SA is an empty INFORMATIONAL response. The response satisfies the request/response pairing within IKEv2 and ensures that a peer is certain that its peer has deleted the SA.

In IKEv2, deleting a pair of IPsec Security Associations and an IKEv2 SA can take up to six packets: a pair to delete the inbound IPsec Security Association, a pair to delete the outbound IPsec Security Association, and a pair to delete the IKEv2 SA.

IPsec Security Associations are created in pairs, with one SA being formed in each direction. When an IPsec Security Association is to be removed, both SAs in the pair must be deleted. Each endpoint

must delete its incoming SAs and allow the other endpoint to close the other SA in each pair. To delete an IPsec Security Association, an INFORMATIONAL exchange with one or more Delete payloads is sent listing the SPIs (as they would be expected in the headers of inbound packets) of the SAs to be deleted. On receipt of this exchange, the peer must close the specified SA(s).

Configuration Payload Exchange

The IKEv2 protocol allows for configuration parameters to be exchanged between peers. The configuration payload is used to exchange these parameters in a request-response or set-acknowledge fashion. The payload is denoted by CP.

RFC7296 describes an exchange in which a client can gain an IP address from the headend as if it were on a local network, although the use of configuration payload is not just limited to assigning IP addresses; it also allows a client that has securely established a connection to a headend to then receive configuration parameters via IKEv2, which can be used for remote-access style deployments. The term IPsec Remote Access Client (IRAC) is used to denote the client and IPsec Remote Access Server (IRAS) the head server.

The (IRAC) initiator can request various parameters in the IKE_AUTH exchange by including a Config Request (CFG_REQUEST) payload containing the parameters it wishes to receive. The responder (IRAS) can then provide the requested parameter details in a Config Reply (CFG_REPLY) payload. The structure of Config Reply consists of the requested parameter(s) and the value(s) returned by the responder.

In both cases, the Config Payload is always sent prior to the Security Association payload.

Any IKEv2 device also has the ability to send configuration parameters to the peer, which is achieved using a Config Set (CFG_SET) payload. This payload will contain the type of configuration parameters that the device is sending to the peer and the value of the parameter. Assuming that the peer accepts these parameters, it will reply with a Config Acknowledge (CFG_ACK) payload denoting which parameters it has accepted. Table 2-5 outlines the configuration payload attributes.

Value	Attribute Type
0	Reserved
1	INTERNAL_IP4_ADDRESS
2	INTERNAL_IP4_NETMASK
3	INTERNAL_IP4_DNS
4	INTERNAL_IP4_NBNS
5	Reserved
6	INTERNAL_IP4_DHCP
7	APPLICATION_VERSION
8	INTERNAL_IP6_ADDRESS
9	Reserved
10	INTERNAL_IP6_DNS
11	Reserved
12	INTERNAL_IP6_DHCP
13	INTERNAL_IP4_SUBNET
14	SUPPORTED_ATTRIBUTES
15	INTERNAL_IP6_SUBNET
16	MIP6_HOME_PREFIX
17	INTERNAL_IP6_LINK
18	INTERNAL_IP6_PREFIX

Table 2-5 *Configuration Payload Attributes*

Cisco IOS allows for a number of proprietary attributes to be sent within the IKEv2 exchange using the configuration payload. These attributes have a value between 16384 and 32767, being private values.

Dead Peer Detection/Keepalive/NAT Keepalive

IKEv2 allows for a liveness check, which can be enabled should a peer fail to receive any traffic from another peer for a configured amount of time or in a periodic manner. This check is performed by sending a blank (but encrypted) INFORMATIONAL payload; on receipt of this payload the receiving peer will reply with its own blank INFORMATIONAL payload, which will then let the original sending peer know that the session is still active and alive.

The use of keepalives allows for failover when designing redundancy into VPN solutions. With

regard to the configuration of the keepalive timer, thought should be given to how many IKE sessions the host will have and what overlay protocols are running over these. IKE and IPsec sessions are expensive to create and maintain. When routing protocols and multiple IKE sessions are used, the routing protocol running within the overlay network should have more aggressive keepalive timers than the keepalive mechanism used by IKEv2. This then allows for the routing protocol to reconverge without having to pull down the IKE session should connectivity between hosts fail for a short period of time.

It is always prudent to use IKEv2 keepalives, for without these, if connectivity fails between two peers, the IKE SA and associated IPsec Security Associations would still be consuming resources on both peers; however they would no longer be productively used.

If IKEv2 keepalives are enabled in a remote access architecture, ensure that there is sufficient capacity on the VPN gateway. In situations where many remote access clients use the same VPN gateway hardware, if the keepalive time is set to a very aggressive value, then the queue for this traffic can become congested at peak times. For this reason, timers should be tuned to ensure that there is adequate capacity on the headend and monitored to ensure that IKE control plane traffic can be dispatched without issue.

When Network Address Translation (NAT) is used between two IKEv2 peers, the resulting IPsec Security Association, the Encapsulation security Payload packet will be encapsulated within the User Datagram Protocol (UDP port 4500). RFC3715, IPsec-Network Address Translation (NAT) Compatibility Requirements, describes how NAT entries on a gateway will have a finite lifetime. This entry is refreshed when traffic is received on the NAT gateway that matches the NAT entry; for this reason NAT-T keepalives can be sent by peers to keep the corresponding NAT entry refreshed in the NAT gateway. RFC3948, UDP Encapsulation of IPsec Encapsulation Security Payload Packets, describes the use of encapsulating this traffic. These keepalives are not secured in any way and do not contain any protected data. As these packets are not encrypted, they can be seen with sniffing tools such as Wireshark or even with a verbose debug, the indication being a single byte payload with the value 0xFF.

Figure 2-9 shows the format of a UDP header when the NAT-T keepalive is in use. Note the single byte payload of 0xFF.

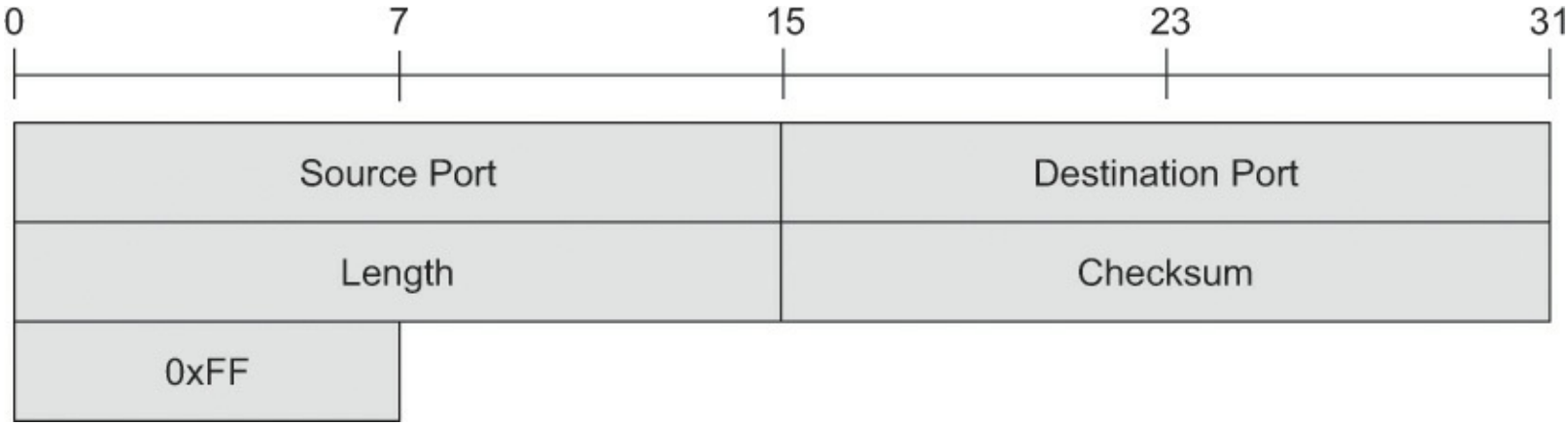


Figure 2-9 *NAT-T Keepalive Packet Format*

Note

Cisco IOS allows for IKEv2 keepalives to be configured using the DPD command . Should the presence of NAT be detected, where the session has floated to using UDP port 4500, if there is a requirement for a keepalives mechanism to maintain the cache in the NAT gateway, then NAT-T keepalives must be used.

IKEv2 Request – Response

All IKEv2 messages are request/response pairs; if a device sends a request, it must receive a response. It is the responsibility of the side sending the request to retransmit if it does not receive a timely response.

The initial exchange usually consists of two request/response pairs. Should DoS protection be required, this will increase the exchange by two pairs. Should the responder not select the Diffie-Hellman group proposed by the initiator, it will be rejected and advised to use another group, resulting in an additional exchange.

If the initiator proposed an invalid group and cookie challenge was enabled on the responder, the IKE_SA_INIT exchange would increase from two exchanges to six.

The IKE_AUTH exchange in its minimal form can consist of a single request-response pair; however, this can grow to many pairs. If EAP is used, then the IKE_AUTH exchange will minimally be six exchanges. RFC7296 states that if EAP is used, IKE_AUTH must support at least 10 exchanges. Each additional CHILD_SA created will incur an additional exchange pair.

IKEv2 and Network Address Translation

The use of Network Address Translation (NAT) exists primarily because of the lack of IPv4 addresses. On January 31, 2011, the last two unreserved IANA /8 address blocks were allocated. To mitigate the need for all Internet-connected devices to require a globally routable IP address, Network Address Translation (NAT) permits an entity to masquerade private network address space with only one publicly routable IPv4 address on the Internet interface of a router on the customer's premises, instead of allocating a public address to each end host.

Nodes that are behind a NAT device have IP addresses that are not globally unique, but rather are assigned from some space that is unique within the network behind the NAT device. These will be commonly allocated RFC1918 private address in the ranges

10.0.0.0/8

172.16.0.0/12

192.168.0.0/16

Figure 2-10 shows two Encapsulation Security Payload packets, the first being sent by a device, which is described as pre-NAT, and the second packet having the source and destination IP addresses changed via a NAT device. Note that UDP encapsulation of Encapsulation Security Payload packets for NAT traversal is not shown in the figure for simplicity.

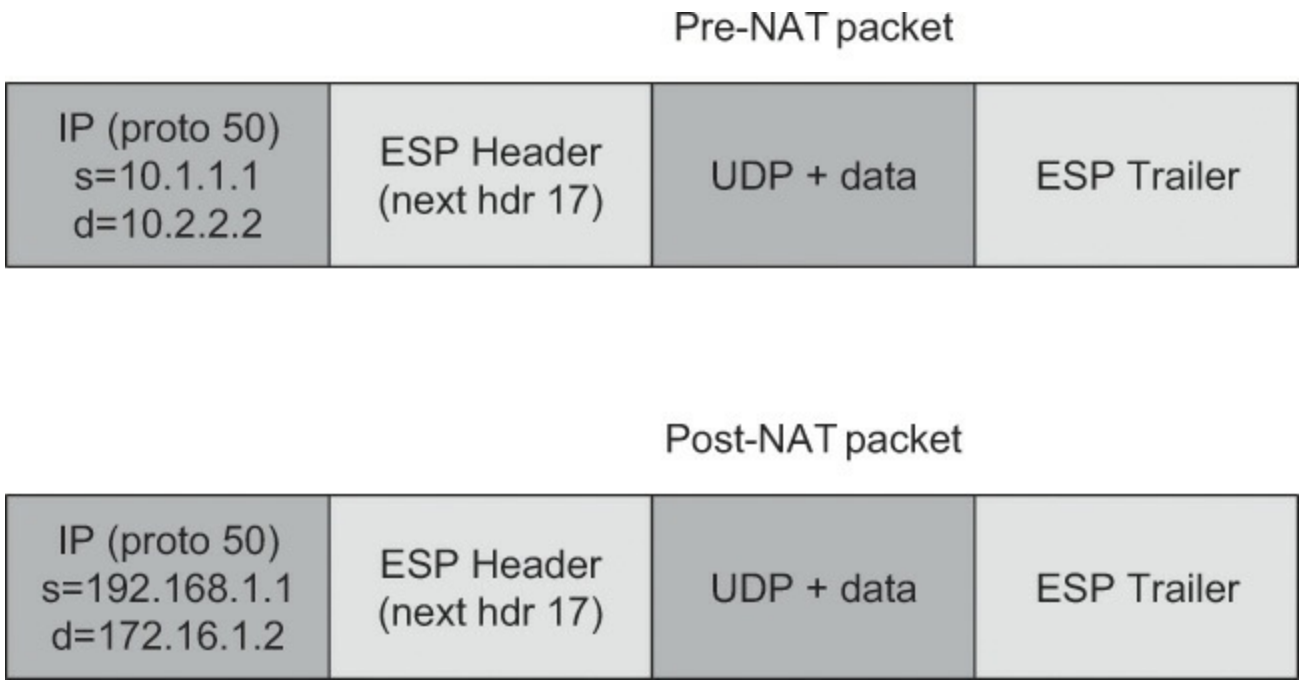


Figure 2-10 *Before and After NAT*

Creating an IPsec connection through a NAT device introduces a number of problems:

If the IPsec connection should use Encapsulation Security Payload with transport mode, and the internal protected data uses either TCP or UDP, then changing the IP address(s) on packets will cause the TCP or UDP checksums to fail, because these have a dependency on the IP source and destination addresses, which are included within the pseudo-header checksum calculation.

If Authentication Header is used, all parts of the packet are integrity protected; when NAT is used on a Authentication Header packet it will result in the received Authentication Header ICV calculation checksums being changed. This checksum is included in the integrity check, so changing it will break the integrity validation. This will occur for Authentication Header in both transport mode and tunnel mode.

When using Encapsulation Security Payload in tunnel mode, a NAT device can struggle to generate a NAT translation because Encapsulation Security Payload is a stateless protocol. It is common for NAT devices to use the transport protocol and ports in the creation of the NAT cache. As Encapsulation Security Payload does not have the concept of ports, like protocols such as UDP or TCP, some NAT devices cannot create a cache for Encapsulation Security Payload traffic. Some NAT devices will use the SPI in the Encapsulation Security Payload header to maintain state for the connectionless setting; however, not all NAT devices have this ability. It is also common for legacy firewall devices to only allow TCP or UDP traffic and deny any other protocol, including Authentication Header or Encapsulation Security Payload.

To overcome the limitations with NAT devices and the use of Encapsulation Security Payload, NAT detection was developed to detect the use of NAT between IPsec peers. If NAT is detected between peers, the IKEv2 and IPsec session would be encapsulated in User Datagram Protocol (IP protocol 17) segments using port 4500, which is denoted in Figure 2-11; this would allow for easy translation by a NAT device.

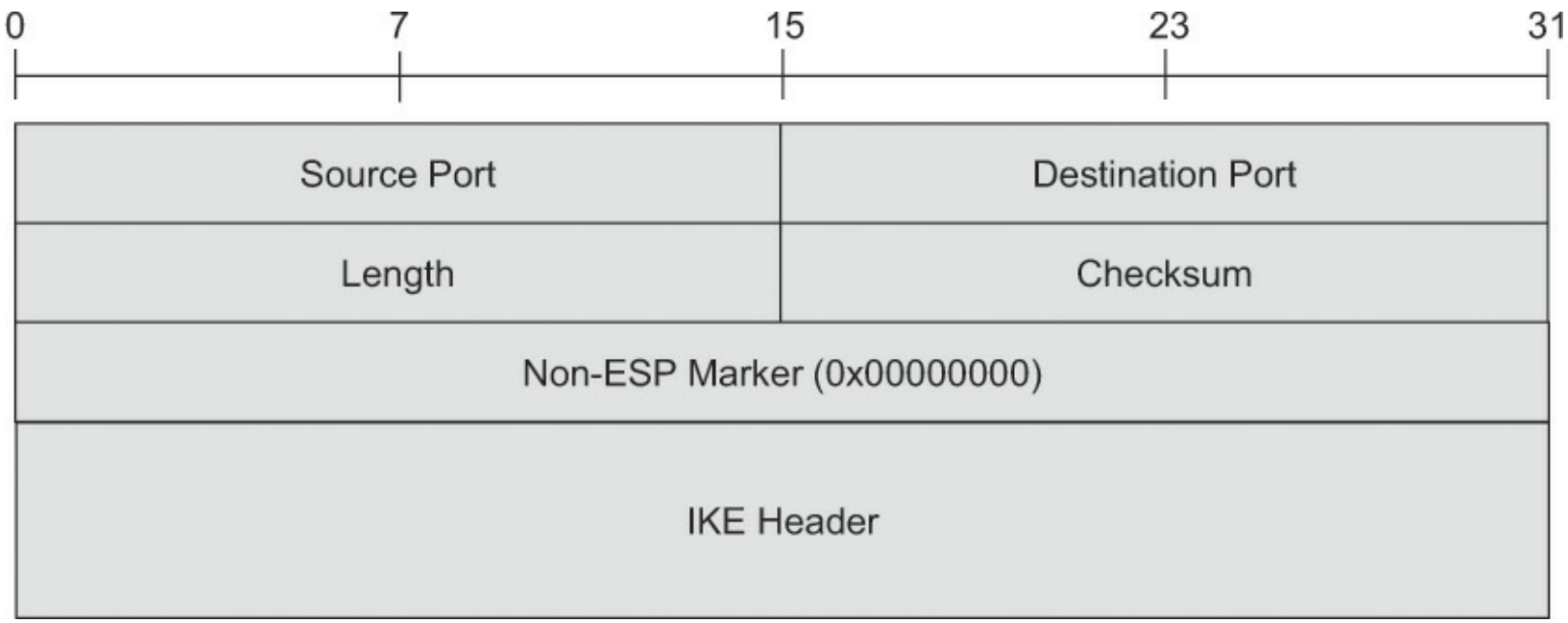


Figure 2-11 *UDP Header for IKE with NAT-T*

When NAT-T is used, both IKE and Encapsulation Security Payload are encapsulated within UDP port 4500. For the receiving host to differentiate between both, there is a method to distinguish Encapsulation Security Payload from IKE. For Encapsulation Security Payload traffic, which is encapsulated within UDP, the UDP checksum will commonly be a zero value.

For IKE traffic there is a 4-byte non-Encapsulation Security Payload marker; this sits where the Encapsulation Security Payload SPI would sit if the traffic were Encapsulation Security Payload. The 4-byte non-Encapsulation Security Payload marker is an all zero value, which indicates that the traffic is IKE.

Figure 2-12 shows the UDP encapsulation of Encapsulation Security Payload, with the differencing fields that identify this from a IKE within a UDP header. The size of a UDP header is 8 bytes.

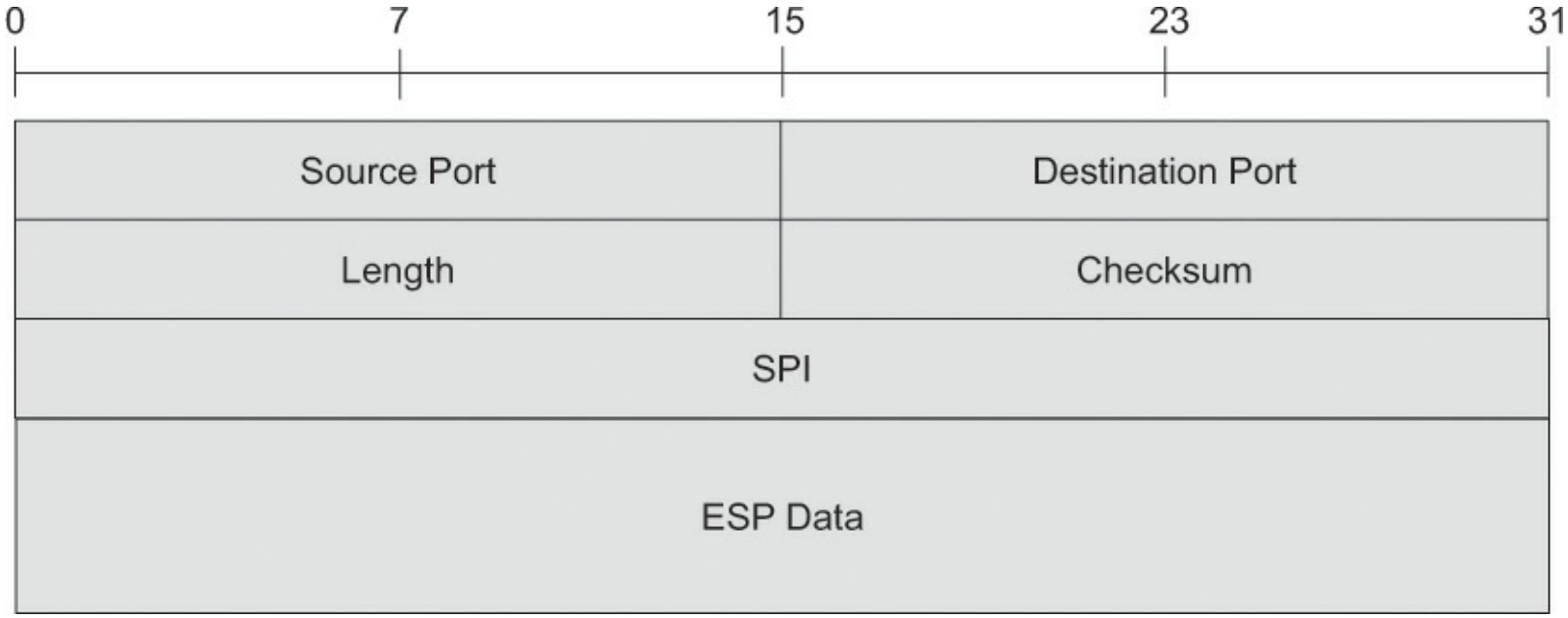


Figure 2-12 *Encapsulation Security Payload Encapsulated within UDP*

Figure 2-13 illustrates the UDP encapsulation when NAT is detected.

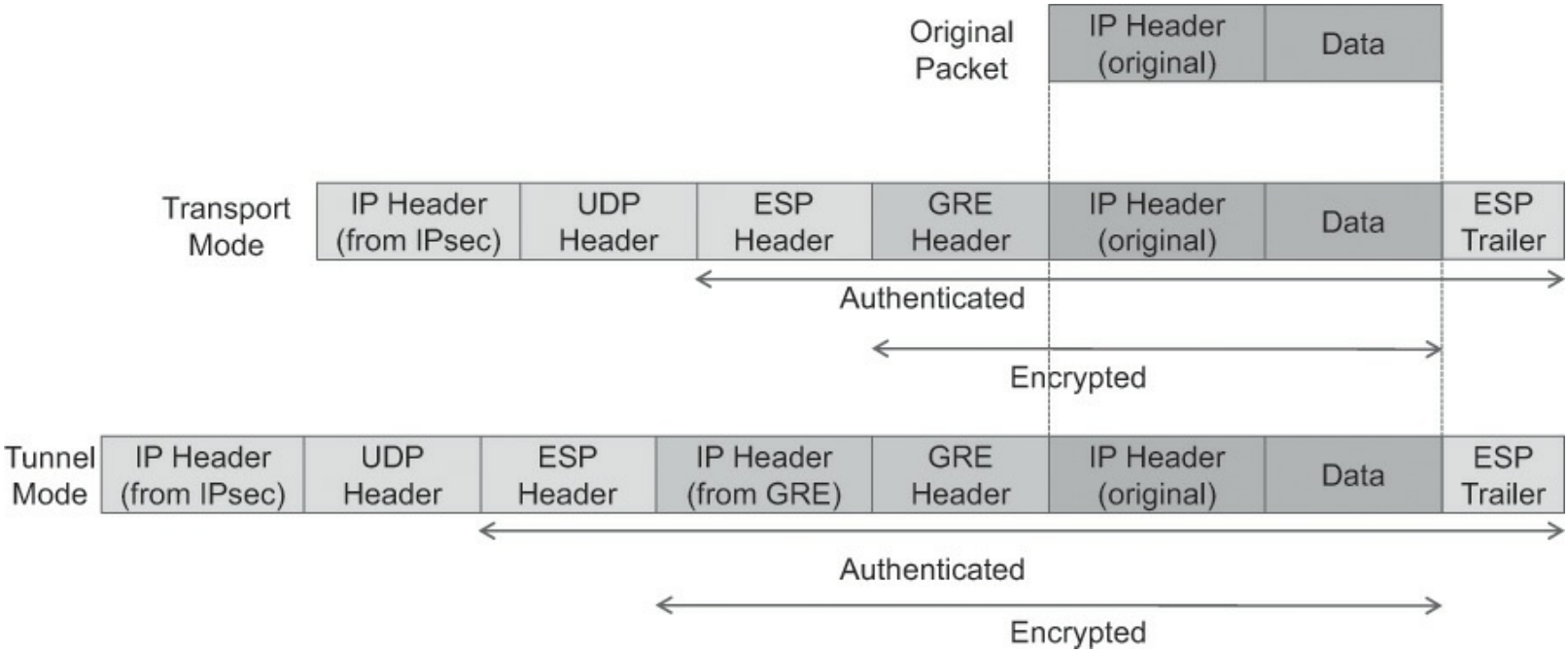


Figure 2-13 *Encapsulation Security Payload with GRE and UDP Encapsulation*

NAT Detection

An IKEv2 client has the ability to include Notify payloads of type NAT_DETECTION_SOURCE_IP and NAT_DETECTION_DESTINATION_IP. These payloads are sent in the IKE_SA_INIT exchange. The NAT_DETECTION_SOURCE_IP payload contains a SHA-1 digest of the SPIs (in the order they appear in the header), the source IP address, and source port. The NAT_DETECTION_DESTINATION_IP payload contains a SHA-1 digest of the SPIs (in the order they appear in the header), the destination IP address, and destination port.

The detection of NAT is also referred as NAT-D, or NAT-Detection, which is described in RFC3947 and RFC3948.

For the initiator, the initiator's SPI will be set and the responder's SPI will not be known and set to all 0's.

As the IP address and port are included in the SHA-1 digest, if the relevant IP address or port is changed between the sender and receiver, this can be detected by calculating the SHA-1 digest of the received attributes and comparing this to the SHA-1 digest included in the relevant notify payload.

Should the NAT-D check fail, that is, if the IP address and or port is changed in transit, the initiator will float the IKE_AUTH exchange from UDP port 500 to UDP port 4500. All IKEv2 and subsequence Encapsulation Security Payload traffic is encapsulated within this UDP tunnel. The use of NAT can be detected by viewing the IKEv2 SA or IPsec Security Associations and noting the port numbers used.

It is not mandatory for an IKEv2 device to support the detection of NAT between hosts; however, it is enabled by default on Cisco IOS. To disable the detection of NAT and the subsequent ability to float to UDP port 4500 should NAT be detected, then the following command can be used:

Click here to view code image

```
no crypto ipsec nat-transparency udp-encapsulation
```

Additions to RFC7296

Although when IKEv2 was originally developed as an RFC it contained a combination of information contained with previous separate RFCs, like most Internet protocols it was still not totally complete. Over time additional RFCs have been written to compliment IKEv2 and bring in additional functionality.

The sections that follow describe complimentary RFCs to date, although this list will probably grow in the future;

RFC5998 An Extension for EAP-Only Authentication in IKEv2

RFC5998 describes the use of the Extensible Authentication Protocol (EAP) to authenticate both the initiator and the responder in an IKEv2 exchange. RFC7296 states that if EAP is used as a means of authentication, then the responder must use a PKI-based authentication. RFC5998 gives direction on how both peers can support the ability to authenticate each other using EAP where no method of PKI authentication is required.

Cisco IOS supports the ability for dual authentication using aggregate authentication, which is described in Chapter 9, "FlexVPN Server."

RFC5685 Redirect Mechanism for the Internet Key Exchange Protocol Version 2 (IKEv2)

RFC5685 describes the ability for an IKEv2 headend to redirect clients to another IKEv2 headend. This is achieved by clients sending the REDIRECT_SUPPORTED notify payload to indicate that it supports this ability. When a headend receives this, it can redirect the client to another headend using a REDIRECT payload containing the address or FQDN of the IKEv2 headend. The IKEv2 load balancer utilizes this technology along with a proprietary protocol (Cisco Load Balancer) to communicate load between IKEv2 headends and make the redirect decision based on the least-loaded headend.

RFC6989 Additional Diffie-Hellman Tests for the Internet Key Exchange Protocol Version 2 (IKEv2)

RFC6989 describes a number of checks that should be applied to the Diffie-Hellman key agreement protocol used within the Internet Key Exchange (IKE) protocol. These checks are advised for implementations that reuse Diffie-Hellman values between sessions; they mitigate an issue where an attacker can send a malformed Diffie-Hellman value that could allow an attacker to efficiently determine the peers' private Diffie-Hellman value. For example, if a VPN headend could reuse the same Diffie-Hellman public values between two clients, client-a and client-b. Without these checks, client-b could send a crafted Diffie-Hellman public value that would allow it to discover the private value used for the session to client-a, and subsequently decrypt any traffic between the headend and client-a when this value was used.

RFC6023 A Childless Initiation of the Internet Key Exchange Version 2 (IKEv2) Security Association (SA)

RFC6023 describes the creation of an IKEv2 SA without also attempting to create a Child SA. This allows IKEv2 to be used solely for Authenticating peers and not for creating IPsec Security Association's. At the time of writing Cisco IOS did not supported this behaviour.

Summary

IKEv2 as a protocol performs some basic high-level functions, such as mutual authentication of peers, negotiation of cryptographic algorithms, creation of a shared secret, and creation and maintenance of IKEv2 and IPsec Security Associations. This chapter describes how this is actually achieved and what happens internally in any implementation at the RFC level.

You should now be familiar with the various packet exchanges within IKEv2, how the initial IKE_SA_INIT exchange is used to negotiate and secure the IKEv2 session, and how authentication then occurs in IKE_AUTH, along with the subsequent creations of IPsec Security Associations.

IKEv2 sessions are maintained using INFORMATIONAL exchanges, which also can be used to remove an IKEv2 SA and to support keepalive functions if required. There will always be an acknowledged reply to any message sent.

The chapter examined what happens when Network Address Translation is used and how IKE detects the presence of NAT within service provider networks and why encapsulating Encapsulation Security Payload within UDP brings a number of benefits.

The packet layout was covered, including how to determine if an IKEv2 packet is from an initiator or responder, how SPIs can be used to track sessions, and how the packet format is structured.

IKEv2 has a number of complementary RFCs that have been developed, adding additional functionality.

References

RFC4945 The Internet IP Security PKI Profile of IKEv1/ISAKMP, IKEv2, and PKIX https://tools.ietf.org/html/rfc4945

RFC7296 Internet Key Exchange Protocol Version 2 (IKEv2) http://tools.ietf.org/html/rfc7296

RFC4739 Multiple Authentication Exchanges in the Internet Key Exchange (IKEv2) Protocol https://tools.ietf.org/html/rfc4739

Chapter 3. Comparison of IKEv1 and IKEv2

This chapter describes the history of the Internet Key Exchange protocol and examines the differences between the Internet Key Exchange version 1 (IKEv1) and Internet Key Exchange version 2 (IKEv2) protocols. Details are given in the differences of the exchanges and the rationale for the decisions made when differences were introduced between both protocols.

IKEv1 and IKEv2 are incompatible protocols, so you will never have an IKEv1 device communicating with an IKEv2 device. Both protocols achieve the same goals, but in totally different manners. These differences are covered in detail below.

IKEv2 introduces a number of features that are not available in IKEv1, such as the use of next-generation encryption protocols and anti-denial-of-service capabilities, which are described throughout the chapter.

IKEv1 offers some of the authentication methods that are used in IKEv2; however, IKEv1 does not allow the use of Extensible Authentication Protocol (EAP), which allows IKEv2 to be the perfect solution for remote-access termination.

IKEv2 has been designed to be more efficient than IKEv1: the packet exchanges are restructured to be lighter, so fewer packets are exchanged and less bandwidth is needed compared to IKEv1.

At the end of this chapter, you will understand that during the development of IKEv1 as many features were not considered in the initial design; however, these issues are addressed in IKEv2.

Brief History of IKEv1

In 1998 a number of protocols were released via the IETF as Request for Comments (RFCs), relating to secure communication over the Internet. These documents described IPsec and covered architecture, the Encapsulation Security Payload protocol, Authentication Header protocol, encryption algorithms, authentication algorithms, Domain of Interpretation (DoI) and key management.

The following RFCs were published by the IETF:

- RFC2401 Security Architecture for the Internet Protocol
- RFC2402 IP Authentication Header
- RFC2403 The Use of HMAC-MD5-96 within Encapsulation Security Payload and Authentication Header
- RFC2404 The Use of HMAC-SHA-1-96 within Encapsulation Security Payload and Authentication Header
- RFC2405 The Encapsulation Security Payload DES-CBC Cipher Algorithm With Explicit IV
- RFC2406 IP Encapsulating Security Payload
- RFC2407 The Internet IP Security Domain of Interpretation for ISAKMP
- RFC2408 Internet Security Association and Key Management Protocol (ISAKMP)
- RFC2409 The Internet Key Exchange (IKE)
- RFC2410 The NULL Encryption Algorithm and Its Use With IPsec
- RFC2411 IP Security Document Roadmap
- RFC2412 The OAKLEY Key Determination Protocol

The Internet Key Exchange (IKE), described in RFC2409, actually referenced a number of other protocols:

- ISAKMP provides a framework for authentication and key exchange but did not actually define these. ISAKMP was designed to be key-exchange independent, allowing the use of numerous key-exchange mechanisms within the framework.
- OAKLEY describes a number of key exchange mechanisms and the services which were provided by each (perfect forward secrecy for keys, identity protection, and authentication). The OAKLEY protocol is used to establish a shared key with an assigned identifier and associated authenticated identities for the two parties within the exchange.
- SKEME describes a versatile key exchange technique that provides a number of modes that delivers the ability for anonymity, reputability, and quick key refreshment.

The Internet Key Exchange describes a protocol using part of the OAKLEY protocol and part of SKEME in conjunction with ISAKMP to obtain authenticated keying material for use with ISAKMP, and for other security associations such as Authentication Header and Encapsulation Security Payload for the IETF IPsec DOI.

Over time numerous RFCs were created that related to RFC2409 including:

- RFC3715 IPsec-Network Address Translation (NAT) Compatibility Requirements
- RFC3947 Negotiation of NAT-Traversal in the IKE
- RFC3948 UDP Encapsulation of IPsec Encapsulation Security Payload Packets

These three RFCs describe ways to detect the use of Network Address Translation between two IKE peers and then the method to encapsulate the traffic within User Datagram Protocol for better NAT traversal of IKE and IPsec traffic.

- RFC3706, A Traffic-Based Method of Detecting Dead Internet Key Exchange Peers, describes the ability for a liveness check between IKE peers.
- RFC4304, Extended Sequence Number (ESN) Addendum to IPsec Domain of Interpretation (DOI) for Internet Security Association, allows larger 64-bit sequence numbers to be employed on Security Associations.

RFC2409 did not address technologies related to Remote Access; this resulted in a number of Internet Drafts being developed to address the requirement to support Remote Access technologies within the Internet Key Exchange.

- Extended Authentication within IKE (XAUTH)
- Extended Authentication within ISAKMP/Oakley (XAUTH)
- The ISAKMP Configuration Method

However, none of these Drafts achieved RFC status and so had limited uptake by vendors. As these solutions were not standardized implementations, different vendors implemented each feature differently. These non-standard implementations caused interoperability issues between software from different vendors.

RFC4754 IKE and IKEv2 Authentication Using the Elliptic Curve Digital Signature Algorithm (ECDSA).

This RFC describes additional authentication methods using the Elliptic Curve Digital Signature Algorithm.

It soon became apparent that the Internet Key Exchange (IKE) protocol described in RFC 2409 had some limitations and that the number of RFCs that were developed to interact with RFC2409 caused a degree of confusion. When Internet Key Exchange version 2 was developed, these limitations seemed to have been addressed with the lessons learned from developing and implementing IKEv1.

Exchange Modes

This section discusses the exchange modes for both IKEv1 and IKEv2.

IKEv1

In IKEv1, multiple modes could be used to establish secure IKE connectivity and authenticate each peer: Aggressive Mode or Main Mode.

Aggressive Mode consists of a three message exchange. Each message in the exchange is known as AM*x*, where *x* denotes the message number. Odd numbers always come from the initiator and even numbers from the responder. The shorter exchange generally resulted in a faster connection; however, it sacrificed the following: the ability to protect identities, the requirement for the responder to authenticate itself first in the exchange, and allowed an intruder with ability to capture the exchange to perform an offline brute force attack if Pre-Shared-Key authentication was used.

Main Mode consists of six message exchanges, which allow for both peers to create a secure IKE session and authenticate each other. Each message in the exchange is known as MM*x*, where *x* denotes the message number. Odd numbers are always from the initiator and even numbers from the responder.

From a high level, the first pair of messages (MM1 and MM2) negotiate cryptographic ciphers, the second pair of messages (MM3 and MM4) exchange key material, and the third pair of messages (MM5 and MM6) are encrypted and used to prove the identity. Unlike Aggressive Mode, the Identity is protected within the exchange. Main Mode has limited anti-DoS capabilities in the form of cookies; however, the exchange of six messages increases the time to establish a session compared to Aggressive Mode, which lacks any form of DOS protection.

Once the IKEv1 peers have established an IKE session, one or more IPsec Security Association(s) can be created; the method used to establish the IPsec Security Association is known as Quick Mode. Each message in the exchange is known as QM*x*, where *x* denotes the message number. Quick Mode commonly uses a three-message exchange, with the first message (QM1) from the initiator containing the cryptographic algorithms to be used and the traffic selectors to define what will be protected. The second message (QM2) in the exchange comes from the responder and contains the cryptographic algorithms to be used and the traffic selectors to define what will be protected. The third message (QM3) comes from the initiator and essentially just acknowledges the responder's previous message. Note that if the commit-bit is set, which requests confirmation for a message, four messages are used. RFC2408 describes this behavior, where the responder will send an acknowledgement to the initiator (QM4).

Assuming that Quick Mode uses three messages, for IKEv1 to create an IPsec Security Association using Aggressive Mode a total of six messages will be exchanged. If Main Mode is used, nine messages will be exchanged.

Figure 3-1 illustrates IKEv1 exchanges using Main Mode and Aggressive Mode.

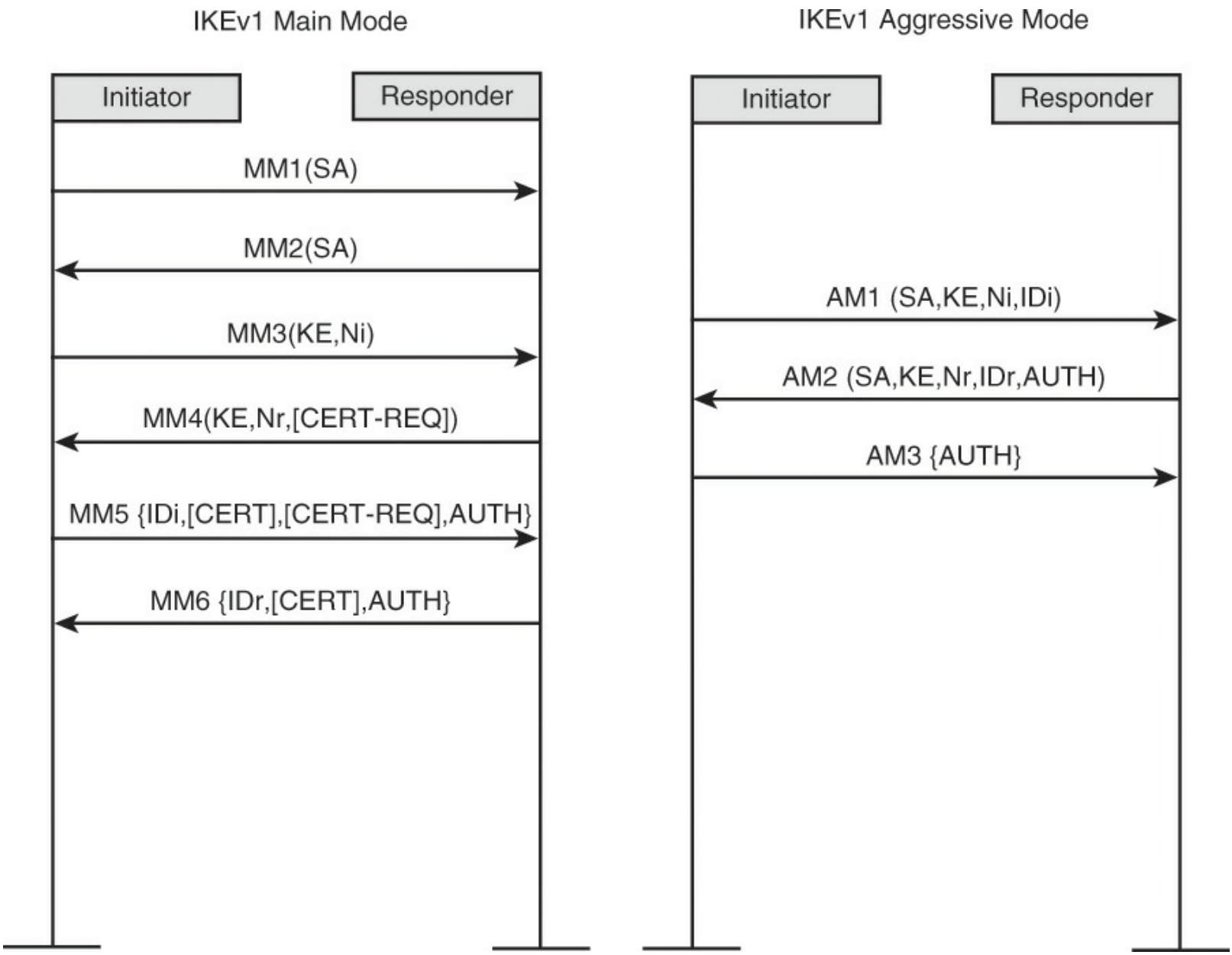


Figure 3-1 *IKEv1 Main Mode and Aggressive Mode*

IKEv2

IKEv2 reduces the number of exchanges from potentially six or nine messages down to four. IKEv2 has no option for either Main Mode or Aggressive Mode; there is only IKE_SA_INIT (Security Association Initialization). Essentially the initial IKEv2 exchange (IKE_SA_INIT) exchanges cryptographic algorithms and key material. So the information exchanged in the first two pairs of messages (MM1 to MM4) in IKEv1 is exchanged in the first pair of messages in IKEv2. The next IKEv2 exchange (IKE_AUTH) is used to authenticate each peer and also create a single pair of IPsec Security Associations. The information that was exchanged in messages MM5 and MM6 of Main Mode and in QM1 and QM2 of Quick Mode is exchanged in the IKE_AUTH exchange, in which both peers establish an authenticated, cryptographically protected IPsec Security Association. With IKEv2 all exchanges occur in pairs, and all messages sent require an acknowledgement (see Figure 3-2). Note if an acknowledgement is not received, the sender of the message is responsible for retransmitting it.

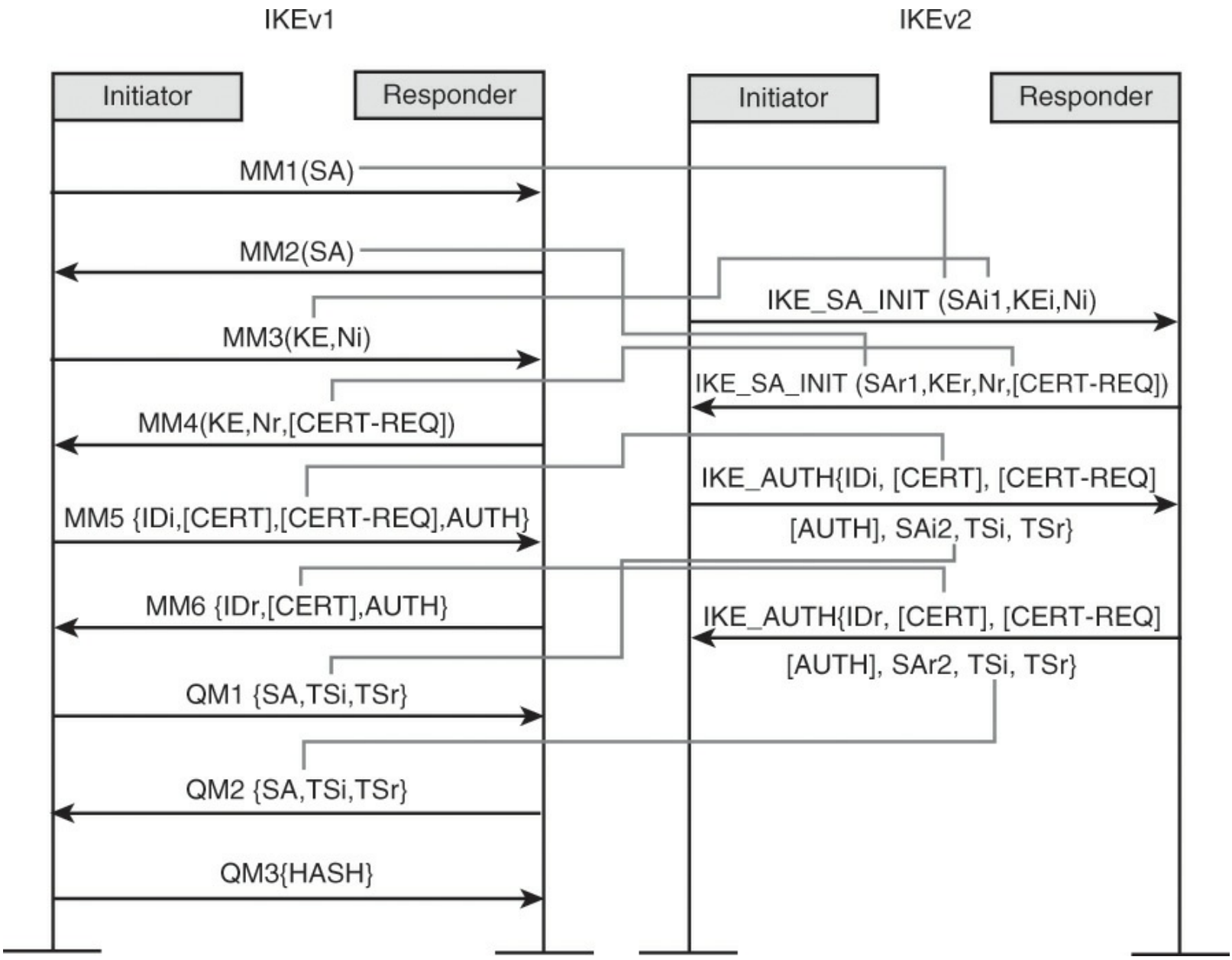


Figure 3-2 *Relationship of Attributes Sent in IKEv1 Exchanges, Compared to IKEv2*

If additional IPsec Security Associations were required in IKEv1, a minimum of three messages would be used by Quick Mode to create these, whereas IKEv2 employs just two messages with a CREATE_CHILD_SA exchange.

Anti-Denial of Service

The reduction in message exchanges within IKEv2 comes at a cost that wasn't a problem within IKEv1. Because cryptographically expensive key material is exchanged in the first two messages (IKE_SA_INIT), an intruder could consume resources on a VPN gateway by sending many IKE_SA_INIT requests with a spoofed source address; as a result the gateway would generate expensive key material and dedicate resource to connections that would not be established. To counter this threat, IKEv2 has the ability to use a stateless cookie, which results in a zero state being assigned to an IKEv2 session if the VPN gateway is under attack.

Lifetime

In IKEv1 the lifetime of the IKE Security Association was negotiated in the first pair of messages (MM1 and MM2). The lifetime configured on the responder must be equal to or less than the lifetime proposed by the initiator for a successful match to occur. This requirement resulted in incompatibilities; IKEv1 sessions would not be established between incorrectly configured peers or peers that had mismatching default lifetimes. In IKEv2 the lifetime is a locally configured value that is not negotiated between peers; a peer will delete or rekey a session when its local lifetime expires. This allows for each IKEv2 peer to control which party will initiate a rekey. This removes any chance that a session will not be established due to a misconfigured lifetime.

Authentication

RFC2409 defined the ability to use a number of methods for authentication:

- Pre-Shared Key (PSK)
- Digital Signature (RSA-Sig)
- Public Key Encryption
- Revised Mode of Public key Encryption

RFC4754 introduced the ability for IKEv1 to use the Elliptic Curve Digital Signature Algorithm (ECDSA) for authentication; however, because this was introduced relatively late in the lifetime of IKEv1, it has seen little acceptance.

IKEv2 allows the use of:

- Pre-Shared Key (PSK)
- Digital Signature (RSA-Sig)
- Extensible Authentication Protocol (EAP)

RFC4754 introduced the ability for IKEv2 to use the Elliptic Curve Digital Signature Algorithm (ECDSA) for authentication; unlike IKEv1, ECDSA was adopted widely in IKEv2, and it is a requirement for implementations to support RFC6380, Suite B Profile for Internet Protocol Security (IPsec).

IKEv2 introduced the ability to authenticate peers via the Extensible Authentication Protocol (EAP), which was described in RFC3748. This allows clients to authenticate via an already established standardized mechanism, unlike IKEv1, which required non-standard extensions. EAP is an authentication framework that allows the use of a number of different authentication methods to validate the identities of peers. Due to its flexibility, it was chosen to be used natively within IKEv2.

Within IKEv1, the authentication method was negotiated as part of the transforms exchanged within MM1 and MM2. Because the authentication method is required to match between both peers, each peer had to support the same authentication method as the device it was connecting to. IKEv2 removes the requirement to negotiate the authentication method and introduces the ability to specify the authentication method in the IKE_AUTH exchange. As a result, each peer is able to choose its method of authentication. This allows for asymmetric authentication to occur, so the peers can use different authentication methods.

High Availability

IKEv1 offers no support for any mechanisms to provide high availability; the only option for high availability is to use tricks such as DNS round robin to load between VPN gateways or to use a dedicated load balancer. IKEv2 provides the ability for a client to be redirected to another VPN gateway. This is documented in RFC5685, Redirect Mechanism for the Internet Key Exchange Protocol Version 2 (IKEv2), which allows any client that supports this ability to be redirected by any compliant IKEv2 VPN Gateway.

Traffic Selectors

The IKEv1 exchange requires a combination of a source IP range, a destination IP range, an IP protocol, a source port, and a destination port per IPsec Security Association. Both parties must agree on the exact traffic selector. However IKEv2 allows for the responder to select a narrower set of traffic selectors than the one the initiator proposed. IKEv2 also allows the use of multiple combinations of a source IP range, a destination IP range, an IP protocol, a source port range, and a destination port range per Child SA. As a result, IKEv2 is able to support both IPv4 and IPv6 traffic for a single IPsec Security Association.

Note that Cisco IOS does not allow both IPv4 and IPv6 traffic to be sent over the same IPsec Security Association.

Use of Identities

IKEv2 supports the ability to generate multiple connections between peers using the same IP address and port pair for the IPsec Security Association, but with the use of different identities. Having the ability to use multiple identities allows IPsec Security Association to be protected by different algorithms and authenticated by a different means.

It should be noted that IKEv2 deliberately allows parallel Security Associations with the same traffic selectors between common endpoints. One of the primary purposes is the ability to use the Differentiated Services Code Point (DSCP) to differentiate traffic between different SAs. This provides the ability to utilize different SAs using different quality of service (QoS) attributes to minimize unnecessary packet drops due to anti-replay check failures.

Network Address Translation

With IKEv1, the ability to detect NAT on the transport network and then act accordingly was included not in the main RFC but in a number of an additional RFCs. IKEv2 included details of how to achieve within the RFC for the protocol itself. IKEv2 also defines how Dead Peer Detection and NAT keepalives are implemented, while these were described by additional RFCs within IKEv1.

Configuration Payload

IKEv1 did not include any method to push or receive configuration data natively, such as a client obtaining an IP address when connecting to a VPN gateway. The ISAKMP Configuration Method is an IETF Draft that allowed for this ability. IKEv2 has the inbuilt ability to support configuration data exchange between peers via the use of a configuration payload.

Mobility & Multi-homing

MOBIKE (IKEv2 Mobility and Multihoming Protocol RFC4555) allows a remote access VPN user move from one IP address to another without reestablishing all security associations with the VPN gateway. The ability to move between IP addresses without performing expensive cryptographic calculations and without the overhead of reconnecting to the VPN gateway improves the user experience. IKEv1 does not support any mechanism to do this, whereas IKEv2 does.

Note that Cisco IOS does not support MOBIKE.

Matching on Identity

For IKEv1 the SKEY (which is used to derive further session keys to protect the exchange) or secret key is composed according to the following formula;

SKEYID = prf(pre-shared-key, Ni_b | Nr_b)

You can see that the pre-shared key is used in the construction of the SKEY; this pre-shared key should be tied to an identity (IP address, FQDN, etc.). When the IKEv1 main mode exchanges are looked at, it is seen that exchanges MM5 and MM6 are encrypted using the SKEY; these packets contain the identity of the peer. As the pre-shared key must be known to decrypt the payload that contains the identity, this makes matching on any identity but IP address impossible when a pre-shared-key is used with IKEv1, as only the IP address is known (as this is the source of the traffic).

IKEv2 implements the SKEY using the following formula;

SKEYSEED = prf(Ni | Nr, g^ir)

Notice that the pre-shared key is not used in this calculation. The only values that are used (the nonces and Diffie-Hellman shared secret) were sent in the IKE_SA_INIT exchange that occurs before the SKEY is required in IKE_AUTH. IKEv2 can use an identity of IP address; however, this identity can be any IP address, not necessarily the source IP address used.

The IKE_AUTH exchange consists of the following;

HDR, SK {IDi, [CERT,] [CERTREQ,] [IDr,] AUTH, SAi2 TSi, TSr} —>

<— HDR, SK {IDr, [CERT,] AUTH, SAr2, TSi, TSr}

The SKEY is used to encrypt the payload within { } and also to decrypt it on the receiver. As the pre-shared key is not used in the SKEY calculation, the peer can decrypt the encrypted payload (using the attributes exchanged in IKE_SA_INIT) and then match the identity. Once the identity is known, the pre-shared key can be selected based on this identity. This key is then used to authenticate the peer.

For the initiator:

AUTH = prf(prf(Shared Secret, “Key Pad for IKEv2”), <InitiatorSignedOctets>)

For the responder:

AUTH = prf(prf(Shared Secret, “Key Pad for IKEv2”),<ResponderSignedOctets>)

IKEv2 removes the constraints of having a unique identity (IP address) per pre-shared key. When a VPN gateway is deployed with IKEv1, if there are multiple separate groups (consumers) of the VPN, each group requires a unique IP address on the VPN gateway to enforce separation so that one group does not have the shared secret of the other group. IKEv2 overcomes this limitation; separate groups can use a VPN gateway with the same IP address while using unique shared secrets. This allows

service providers to reuse hardware for multiple customers at the same time protecting each customer's data.

Note

IKEv1 uses the pre-shared key in the construction of the shared secret, unlike IKEv2, which only uses the derived shared secret from the Diffie-Hellman exchange and the nonces. If an attack became apparent against Diffie-Hellman then the inclusion of the pre-shared key brings a number of security benefits. Using a pre-shared key in IKEv1, where the key is generated with strong entropy, would require an attacker to break not only the Diffie-Hellman exchange but also the pre-shared-key. In IKEv2, an attacker would only need to break the Diffie-Hellman exchange. For this reason at the time of this writing, IKEv1 with a pre-shared key is known to be resistant to attacks using quantum computing, or quantum resistant, but IKEv2 is not, at least not with the algorithms currently used. However the following IETF Draft describes a method to enable IKEv2 to act in a quantum-resistant manner: An Extension for Postquantum Security using Preshared Keys for IKEv2 draft-fluhrer-qr-ikev2-00.

This is purely speculative and is based on the question; what would happen if a quantum computer was ever made that could break Diffie-Hellman?

Reliability

All message exchanges within IKEv2 are always sent in pairs, consisting of a request and response message; if the sender of the request message does not receive a response, it will retransmit the request. Because all messages are in pairs and all messages must be acknowledged, all exchanges are reliable.

Cryptographic Exchange Bloat

Within IKEv1 all sets of cryptographic ciphers that are negotiated to protect the IKE session are transferred in separate transforms, so if a device was required to support multiple sets of cryptographic algorithms, it would send a separate transform for all the sets of cryptographic algorithms that it supported. Within IKEv2 all cryptographic algorithms are sent within one transform (or two transforms if combined and non-combined mode ciphers are used), which can greatly reduce the amount of traffic sent when negotiating and rekeying IKEv2 Security Associations.

Combined Mode Ciphers

IKEv1 uses separate algorithms to provide encryption and integrity protection. With the availability of combined mode ciphers, a new class of algorithms was introduced, in which a single algorithm can provide both encryption and integrity protection. IKEv2 provides the ability to use these ciphers; however, IKEv1 is not able to support these as they are not defined within the IANA assignments after being removed from RFC4869 (Suite B Cryptographic Suites for IPsec), after this was updated by RFC6869.

Continuous Channel Mode

IKEv2 introduced the concept of continuous channel mode. In it, once a new IKE Security Association is created, the new IKE SA inherits all the valid child Security Associations that belonged to the old IKE SA. These child Security Associations can be accessed from both the old and new SAs until the old SA expires. In addition deleting an IKEv2 SA will delete any associated IPsec Security Associations. IKEv1 operates in a noncontinuous channel mode, in which if a new IKE SA is generated, all corresponding IPsec Security Associations will be generated also. In IKEv1 an IPsec Security Association could exist without an IKEv1 SA, but in IKEv2 this is not possible, since the IKEv2 SA must exists for the IPsec Security Association to exist.

Summary

Internet Key Exchange version 1 has some limitations; like many protocols, its deficiencies were not discovered until implementations interoperated with each other. Internet Key Exchange version 2 was developed to overcome the limitations discovered within Internet Key Exchange version 1. Currently, there is little (to no) development within the IP Security Maintenance and Extensions (ipsecme) working group for IKEv1, whereas IKEv2 is actively being discussed and developed.

There are many benefits to using IKEv2 over IKEv1; especially the security that IKEv2 brings in the way of cryptographic algorithm support and its ability to implement anti-DOS capabilities should a number of half-open connections be detected.

IKEv2 is far more efficient than IKEv1; it requires fewer messages to establish an IPsec Security Association and is able to send all available cryptographic algorithms in a single transform, unlike to IKE1, which requires a separate transform per set of cryptographic policies.

IKEv2 is very well suited to remote access solutions. It has features such as the ability to support MOBIKE and match on identity when using pre-shared key authentication that are not available in IKEv1.

As you have learned, there are many reasons to use IKEv2, and at the time of this writing the majority of new customer deployments are using IKEv2 instead of IKEv1 as the benefits are clear. The majority of third-party clients are now supporting IKEv2 and moving away from using IKEv1.

References

http://tools.ietf.org/html/draft-ietf-ipsec-ikev2-tutorial-01 Understanding IKEv2: Tutorial, and rationale for decisions

https://tools.ietf.org/html/rfc6071 IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap

http://www.iana.org/assignments/ipsec-registry/ipsec-registry.xhtml Internet Key Exchange (IKE) Attributes

http://datatracker.ietf.org/wg/ipsecme/documents/ IP Security Maintenance and Extensions (ipsecme)

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.78.1517&rep=rep1&type=pdf IKEv1 and IKEv2: A Quantitative Analyses

https://tools.ietf.org/html/draft-dukes-ike-mode-cfg-00 The ISAKMP Configuration Method

https://tools.ietf.org/html/draft-fluhrer-qr-ikev2-01 Postquantum Preshared Keys for IKEv2

Part III: IPsec VPNs on Cisco IOS

Chapter 4. IOS IPsec Implementation

The following chapter reviews the modes of encapsulation used within IPsec by describing tunnel and transport modes and how these are used. If you have not read chapter 1, "Introduction to IPsec VPNs,"—which describes tunnel and transport mode, it may be prudent to do so before starting this chapter.

Before tunnel interfaces are described, crypto map should be discussed. A crypto map is a Cisco IOS software configuration attribute that performs a number of functions related to setting up an IPsec Security Association. Within a crypto map, data flows are defined that are to be protected by the SA and require security processing or traffic that is required to be bypassed; these are referenced with an access control list. The way that the traffic will be protected (which security policy, protocol, mode, and algorithms) is defined by the IPsec transform-set, and to whom the session will be established is defined by the peer statement. These attributes can be aligned to the Security Policy Database (SPD).

A crypto map is applied to an interface, which then enables the SPD instance; this results in data flows that have the correct policy applied. A crypto map can be applied on a main/physical or tunnel interface (with certain restrictions) and performs IPsec protection before interface encapsulation.

As an alternative to a crypto map, Cisco IOS also allows a tunnel interface with tunnel protection. This is a logical interface that represents the source and destination endpoints of the tunnel. The tunnel interface interconnects the transport network (the network that the tunnel source and destination will use) and the overlay network (the traffic that will transverse within the tunnel). The tunnel interface allows various forms of encapsulation (defined as *modes*); although we will focus on generic routing encapsulation (GRE) or IPsec as the modes of encapsulation, other modes can be used, such as IP-in-IP, Distance Vector Multicast Routing Protocol (DVMRP), or IPv6-in-IPv4. However, these will not be discussed, for they are not relevant to using IKEv2 or IPsec.

Generic routing encapsulation (GRE) is covered, as well as the reasons why this is needed when using non-IP protocols or multicast. The overhead that is incurred by GRE is discussed, along with its benefits, such as mixed mode and dual stack for passing both IPv4 and IPv6 over a single IPsec Security Association.

The two modes that are relevant within this chapter and the remaining book are tunnel mode GRE and tunnel mode IPsec: both of these modes allow for the tunnel to become protected (by IPsec) by applying what is known as tunnel protection.

Tunnel protection can be applied only to a tunnel interface. The protection profile specifies the security policy that is applied to the IPsec Security Association, including how to protect, the protocol, the mode (transport or tunnel) and the cryptographic algorithms used. Unlike crypto maps, tunnel protection performs IPsec protection after interface (tunnel) encapsulation.

To highlight the differences between using a crypto map and tunnel protection, we can take two examples: the first uses tunnel protection, where the tunnel protection uses Encapsulation Security Payload in transport mode and is applied to a GRE tunnel interface. Traffic that passes into the tunnel is encapsulated within GRE and is then encrypted by the tunnel protection.

Figure 4-1 illustrates the behavior when a tunnel interface with tunnel protection is used. Note that since the encapsulated packet will be IP, the protocol type within the GRE header, which denotes the encapsulated traffic, is described with the ethertype value (0×0800), which is IPv4.

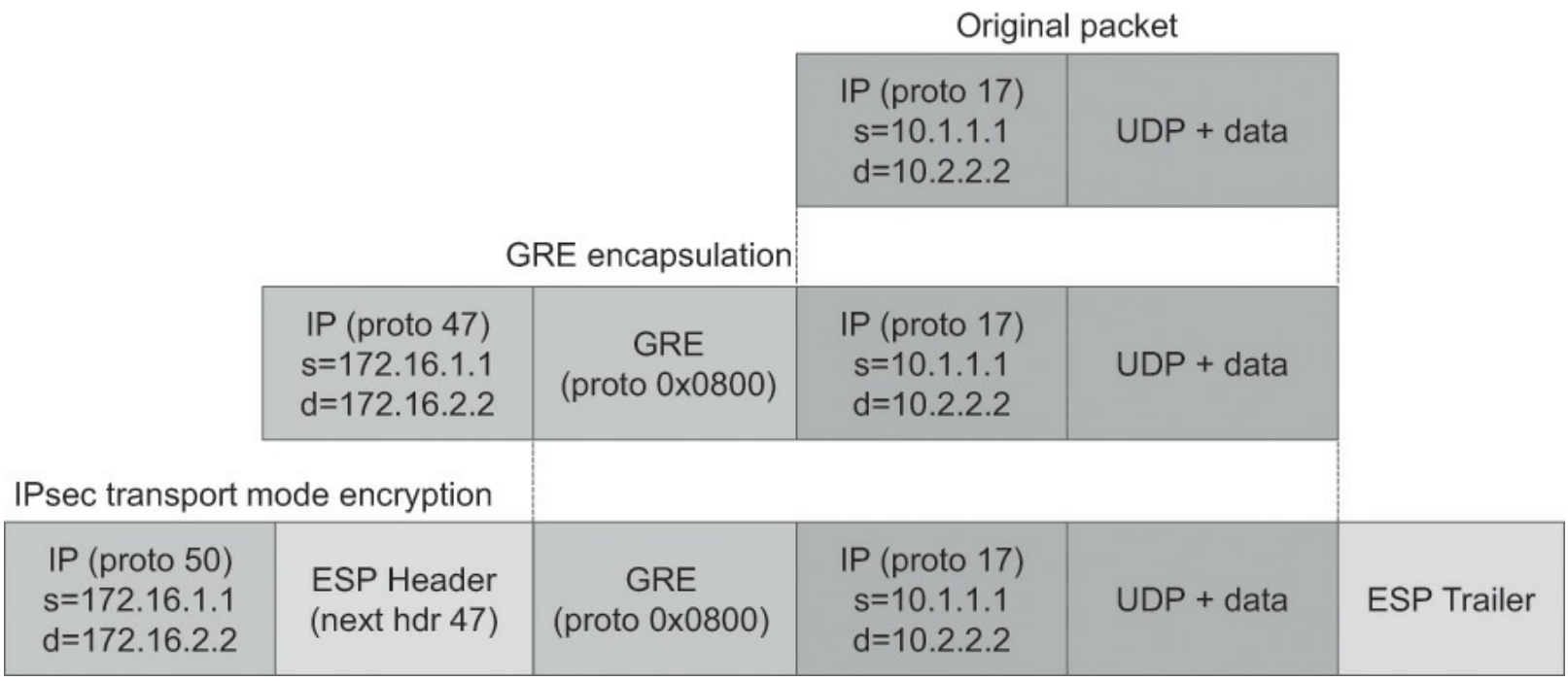


Figure 4-1 *GRE Encapsulation with Tunnel Protection*

In contrast, if a crypto map is applied to a GRE tunnel interface (with no tunnel protection), we see that traffic is protected by the crypto map as soon as it enters the tunnel interface; because tunnel mode is used, the entire packet is encrypted. This traffic is then GRE encapsulated, with the outer IP headers being composed of the GRE tunnel source and destination addresses. Note that in Cisco IOS version 15 code and later this is an invalid configuration on Cisco IOS and will result in an error when the crypto map is applied to the tunnel interface. Before Cisco IOS version 15 code it was possible to apply a crypto map to a tunnel interface, but by default tunnel mode would also be enabled, rather than transport mode (even when the IPsec transform-set used transport mode). This is because RFC4301 has the strict requirement that an IPsec gateway uses transport mode only for locally sourced traffic. This example is included only to illustrate the behavior when using crypto maps.

Figure 4-2 illustrates the contrasting behavior, where you can see that the original payload is encrypted and then GRE encapsulated.

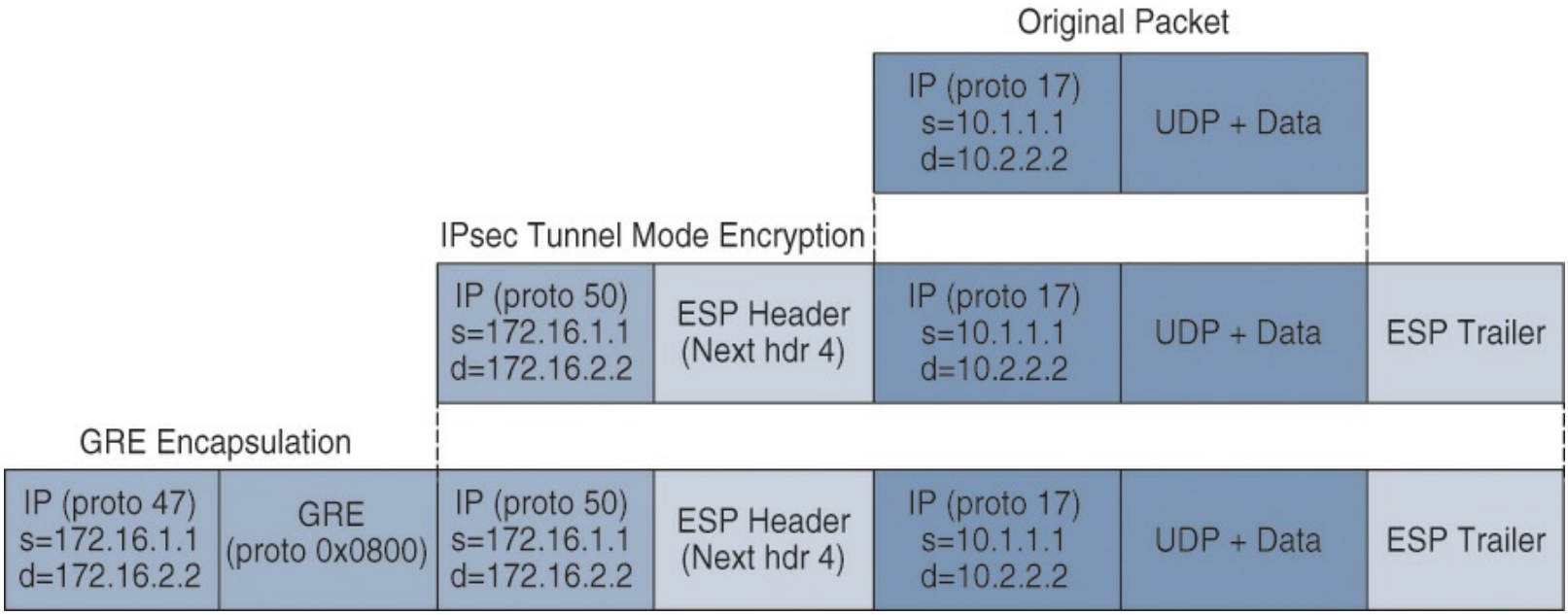


Figure 4-2 *Encapsulation Using Crypto Map Applied to a GRE Tunnel Interface*

You will learn why Cisco moved from using legacy crypto maps, which will help you understand why the VPN architecture was changed on Cisco IOS with the introduction of tunnel interfaces. After reading this chapter, you will be aware of the different types of tunnel interfaces, which will make you able to know where to position Virtual Tunnel Interfaces (VTI) or GRE Tunnel Interfaces in a static or dynamic model.

For each interface type, the chapter describes how traffic is encapsulated and the effect that this has on the route lookup; you will learn how packets move through the encapsulation/decapsulation engines and what effect this has on the route lookup.

The way to enable IPsec on the interface is described, with the various tunnel encapsulation modes. Static and dynamic tunnel interfaces will be covered, which will let you understand where to position each.

Virtual Routing and Forwarding (VRF) is introduced, along with the ability of limiting passing traffic between VRFs using tunnel interfaces.

This chapter introduces command line interface outputs to help you understand the technologies when virtual private networks are configured on Cisco IOS and IOS-XE. In the rest of this chapter, IOS will imply IOS and IOS-XE.

Modes of Encapsulation

The method of encapsulation is defined by the mode of the tunnel; the following section describes GRE when using IPsec tunnel and transport mode.

GRE Encapsulation

Generic routing encapsulation (defined by RFC2784 and extended by RFC2890) provides a simple mechanism to encapsulate packets of any protocol (the payload packets) over any other protocol (the delivery protocol) between two endpoints. GRE places its own header (4 bytes plus options) between the payload packet and the delivery header. Figure 4-3 shows the structure of the GRE header.

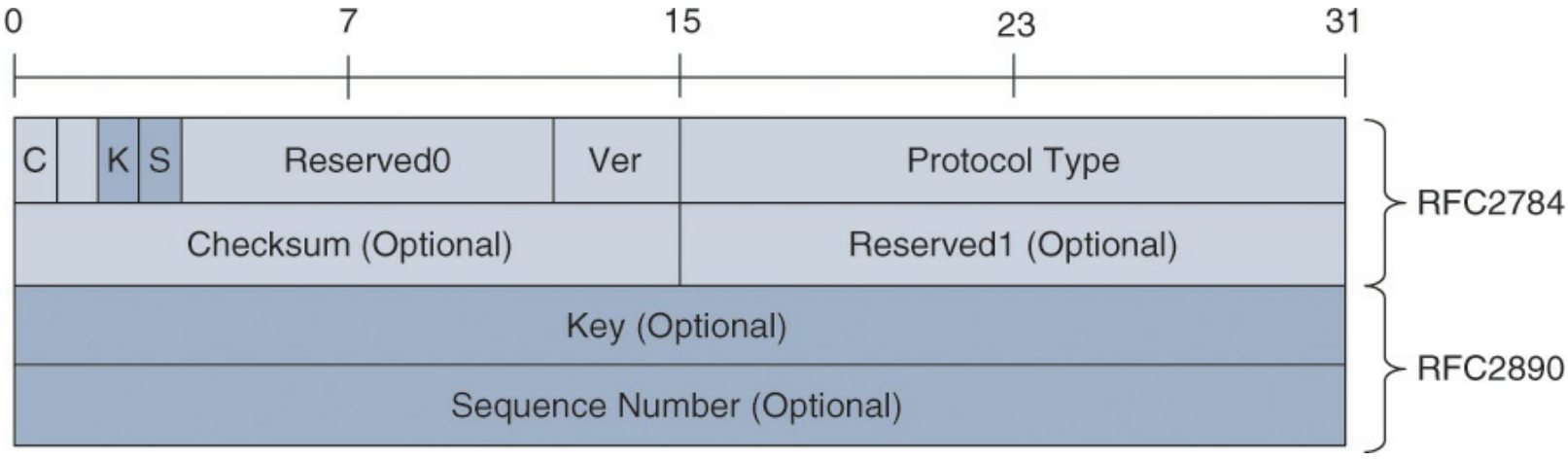


Figure 4-3 *GRE Header*

The C bit indicates the presence of the 16-bit (2-byte) Checksum field (one's complement of the GRE header and payload) and 16 bits (2 bytes) of padding (Reserved1). The K bit is set if the 32-bit (4-byte) Key field is present; it allows for multiplexing traffic flows across GRE tunnels and is often used to differentiate traffic sourced from or destined to a specific tunnel interface. The S bit is set if the 32-bit (4-byte) Sequence Number field is present; it enables in-order delivery of packets.

A common case is GRE over IPv4 or IPv6 (GRE/IP for short). GRE/IP encapsulation results in a new IP packet being generated on the router, with the local GRE endpoint IP address as the source and the remote GRE endpoint IP address as the destination. The protocol number in the outer IP header is set to 47 (0×2F). The outer IP header is either IPv4 or IPv6, depending on whether the endpoint addresses are defined as IPv4 or IPv6.

With GRE/IP, the overhead compared to the original packet is:

- 4 bytes (+ GRE options) for the GRE header,
- 20 bytes (+ IP options) for the outer IPv4 header (GRE/IPv4), or
- 40 bytes (+ extension headers) for the outer IPv6 header (GRE/IPv6).

Figure 4-4 shows the encapsulation of a UDP over IPv4 packet into GRE/IPv4. The UDP traffic endpoints are 10.x.x.x, and the GRE tunnel endpoints are 172.16.x.x.

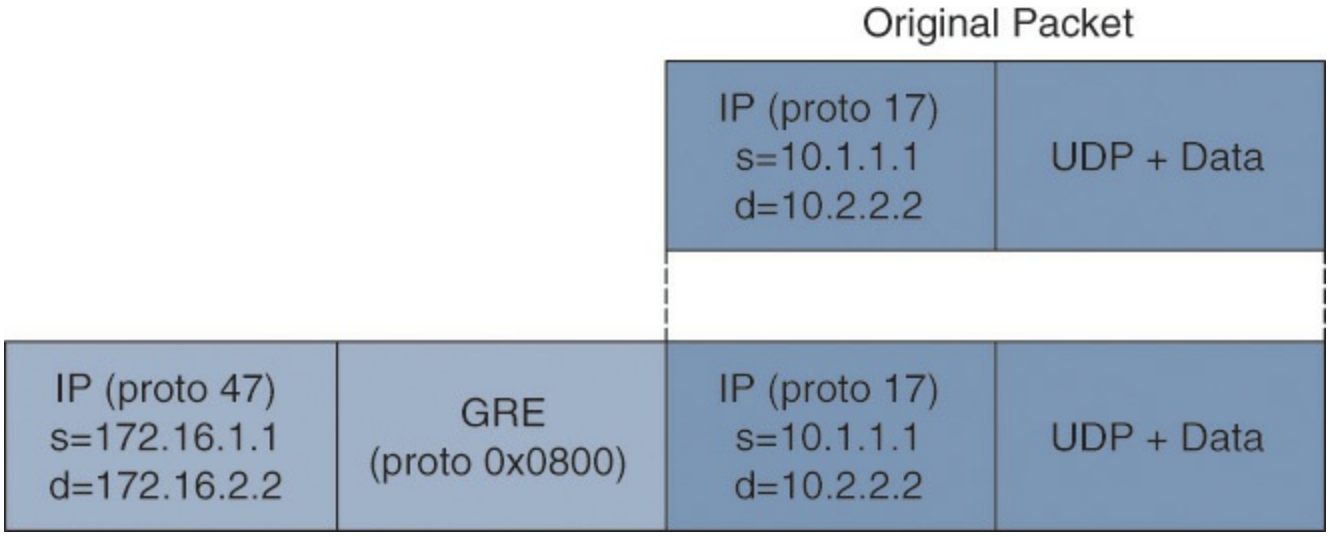


Figure 4-4 *GRE Encapsulation*

A later subsection in this chapter, "Multipoint GRE," explains the concept of GRE-based multipoint interfaces (known as mGRE). A common misconception is that the mGRE overhead is 28 bytes, as opposed to 24 bytes for plain GRE overhead. In fact there is no such thing as mGRE overhead; the misconception comes from the frequent use of a GRE tunnel key with a multipoint interface for multiplexing purposes.

GRE over IPsec

In GRE over IPsec (usually written GRE/IPsec for short), data packets are first encapsulated within GRE/IP, which results in a new IP packet being created inside the router. This packet is then selected for encryption (the traffic selector being GRE from local to remote endpoint IP address), and encapsulated into IPsec. The actual encapsulation depends on whether tunnel or transport mode is used. Since a new IP header has already been added, IPsec transport mode is generally used to keep the overhead to a minimum (with it there is no need to add another 20-byte IP header); however this is possible only if the GRE and IPsec traffic terminate on the same endpoints.

IPsec Transport Mode with GRE over IPsec

With GRE over IPsec transport mode, a plaintext IPv4 or IPv6 packet is encapsulated in GRE/IP and then protected by IPsec for confidentiality and/or integrity protection; the outer IP header with tunnel source and destination addresses helps route the packet correctly.

The GRE over IPsec transport mode overhead compared to the original packet is by default the GRE plus the crypto overhead, along with the outer IPv4 or IPv6 header, which is the tunnel source and destination (see chapter 15, "IPsec Overhead and Fragmentation," for more details).

Figure 4-5 shows the encapsulation of a UDP over IPv4 packet using GRE over IPsec transport mode. The UDP traffic endpoints are 10.x.x.x, and the IPsec tunnel endpoints are 172.16.x.x. The protocol field in the outer IP header is set to 50 (0×32) for Encapsulation Security Payload.

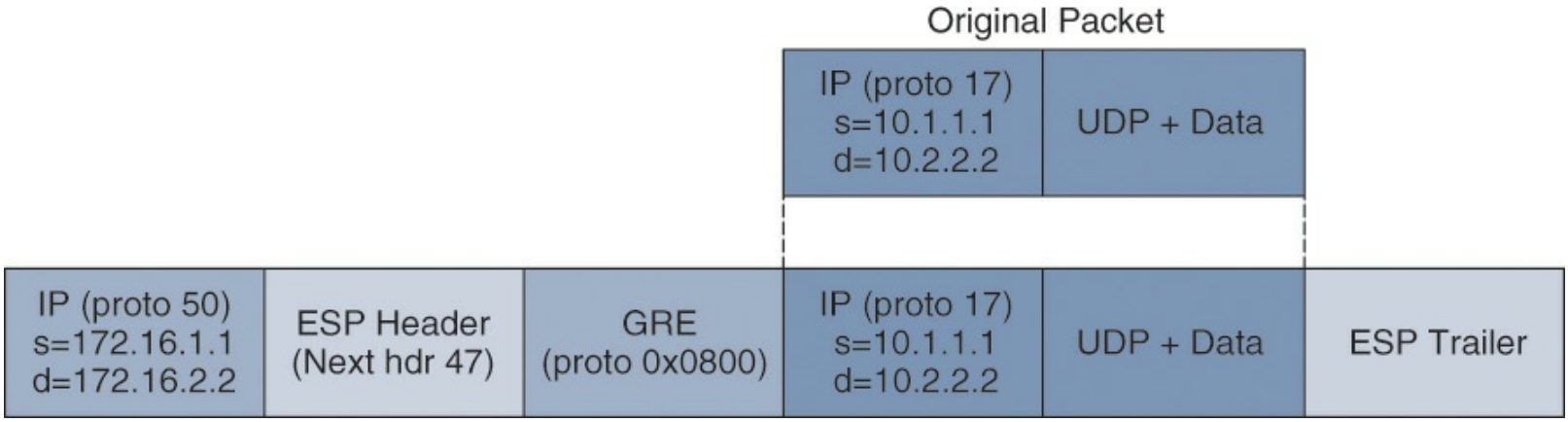


Figure 4-5 *GRE over IPsec with Transport Mode*

IPsec Tunnel mode with GRE over IPsec

With GRE over IPsec tunnel mode, a plaintext IPv4 or IPv6 packet is encapsulated into GRE and then into another IPv4 or IPv6 packet containing the tunnel source and destination IP addresses. This is protected by IPsec for confidentiality and/or integrity protection, with finally an additional outer IP header being used as the tunnel source and tunnel destination to route the traffic to the destination.

The IPsec tunnel mode overhead compared to the original packet is:

- 40 bytes (more if IP options are present) for the outer and inner IPv4 headers, or
- 80 bytes (more if extension headers are present) for the outer and inner IPv6 headers, plus the GRE (4 byte) and crypto overhead (see chapter 15, "Computing the IPsec Overhead").

Figure 4-6 shows the encapsulation of a UDP over IPv4 packet into Encapsulation Security Payload tunnel mode with GRE transport. The UDP traffic endpoints are 10.x.x.x, and the IPsec tunnel endpoints are 172.16.x.x. The protocol field in the outer IP header is set to 50 (0×32) for Encapsulation Security Payload. Note that since the encapsulated packet will be IP, the protocol type within the GRE header, which denotes the encapsulated traffic, is described with the ethertype value (0×0800), which is IPv4.

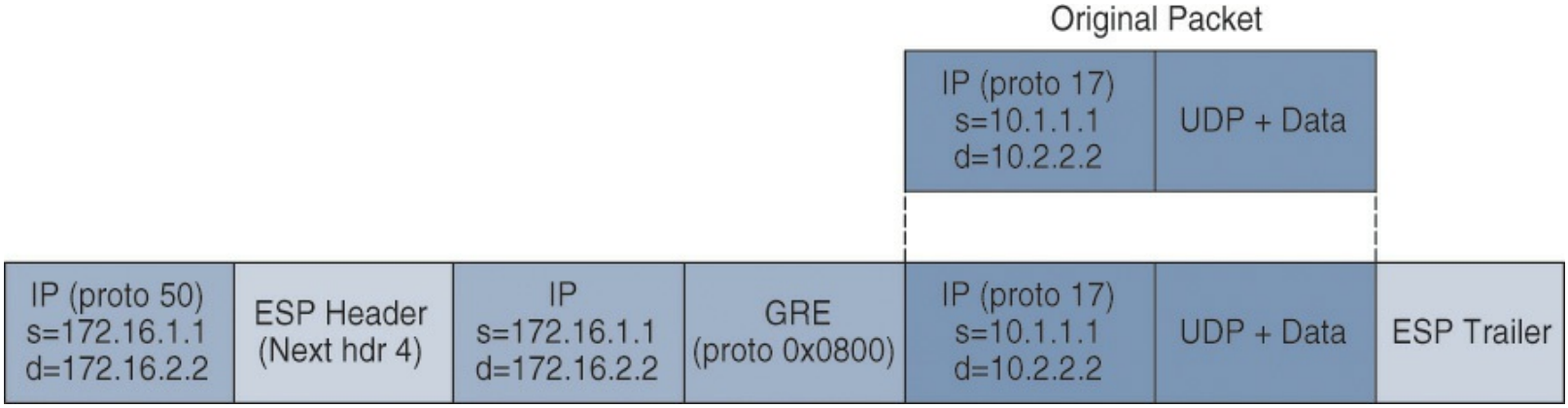


Figure 4-6 *GRE over IPsec with Tunnel Mode*

It should be noted that the inner IP header, which is placed after the Encapsulation Security Payload header and before the GRE header, is encrypted and can't be used in traffic routing decisions. This header is really redundant and serves little useful purpose. The only case when the inner IP header is needed, and so GRE over IPsec tunnel mode is used, occurs when DMVPN is used with mGRE and multiple spokes are behind the same PAT device. In this case the inner IP header can be used to differentiate the spokes. It should be noted that this limitation does not occur when using IKEv2 with virtual-access interfaces, as they are inherently point-point. For this reason GRE/IPsec in transport mode should be used where possible to save of 20 bytes overhead for IPv4 and 40 bytes for IPv6.

Traffic

There are a number of limitations with the traffic that can be protected when using encapsulation modes and their method of protection. The following sections will describe the limitations and benefits of the various methods of protection.

Multicast Traffic

When multicast is used, it is possible that if a device has multiple peers and as a result has security associations from multiple sources, these can conflict with each other. For multicast traffic, there is the issue of multiple destination systems associated with a single SA.

For this reason, a method or system is required to coordinate among all multicast devices the ability to select an SPI or SPIs on behalf of each multicast group and then communicate the group's IPsec information to all of the legitimate members of that multicast group via some mechanism.

When legacy site-to-site crypto maps on Cisco IOS are used, the transport of multicast data traffic over the VPN tunnel will be prevented; however, locally sourced multicast traffic will be permitted, enabling peering across the VPN by using a multicast-based routing protocol (like OSPF or EIGRP, for example).[1]

[1] See Cisco bug CSCtq94342 Self-originated, multicast traffic handling through IPsec tunnel, for details.

Specific standards have been developed for this, such as the Multicast Group Security Architecture (RFC3740) and the Group Domain of Interpretation (RFC6407), and Cisco's GET VPN solution implements and extends those standards. However, this technology introduces specific requirements and constraints in terms of network architecture, such as the use of a private transport network.

Another possibility is to encapsulate multicast into GRE (using the same principle as for mixed mode and dual stack).

Fortunately, virtual IPsec interfaces (known as VTI and described in the next section) do not have this limitation, as their point-to-point nature and any-to-any SA negotiation enable multicast to flow across a pure IPsec tunnel without resorting to GRE. This means that multicast-based routing protocol neighborships as well as multicast data traffic forwarding become possible using native IPsec virtual interfaces.

Non-IP Protocols

In some cases, it is necessary to transport non-IP traffic over an IPsec VPN. Examples include the Cisco Discovery Protocol (CDP) or the Next Hop Resolution Protocol (NHRP) that enables dynamic mesh capabilities in DMVPN and FlexVPN.

Transporting non-IP protocols cannot be achieved using native IPsec, for it is only meant to transport

IPv4 or IPv6 traffic by definition of the traffic selectors. GRE is again the solution to this problem: By encapsulating non-IP protocols with GRE, IPsec can then be used to protect this traffic.

This is why GRE encapsulation is mandatory in dynamic mesh scenarios that rely on NHRP, even though it may seem like native IPsec would be sufficient.

Table 4-1 summarizes the protocols that can be transported over the types of IPsec tunnels mentioned in this chapter (the crypto map row represents IPsec tunnel mode without prior GRE encapsulation and without the use of a VTI).

	IPv4 only	IPv6 only	IPv4 & IPv6	IP Multicast	Non-IP
Crypto map	Yes[1]	Yes[2]	No	No[5]	No
IPv4 VTI	Yes	Yes[3]	No	Yes	No
IPv6 VTI	Yes[3,6]	Yes	No	Yes	No
IPv4 DVTI	Yes	Yes[3]	No	Yes	No
IPv6 DVTI	Yes	Yes	No	Yes	No
GRE/IPsec[4]	Yes	Yes	Yes	Yes	Yes

[1]IPv4 transport only.

[2]IPv6 transport only.

[3]Mixed mode since 15.6.(1)T.

[4]IPv4 or IPv6 transport.

[5]Local multicast traffic supported.

[6]Mixed-mode remote access with AnyConnect since 15.2(3)T.

Table 4-1 *Traffic Permitted by Protection Type*

The Demise of Crypto Maps

The original implementation of IPsec VPNs used on Cisco IOS was known as crypto maps. The concept of configuring a crypto map was closely aligned to the IPsec protocol, with traffic that was required to be encrypted being defined in an access control list. This list was then referenced within an entry in the crypto map along with the IPsec cryptographic algorithms within the transform set.

This configuration could become overly complex, and when long access control lists were used, it was very common for mismatched access control entries to result in the creation of a security association that did not include all traffic that was intended to be protected. As a result, traffic that should have been protected was sent in the clear.

It is also the case that the access control list must be updated to reflect new networks if additional networks must be protected. If a VPN Gateway is using crypto maps and has 100 peers, a new access control entry is required for every remote prefix on the gateway and every peer, resulting in a potential administrative nightmare.

The crypto map is applied to an interface and is activated when traffic is sent or received over the interface. The result is limited flexibility if multiple interfaces are to be used or if for some reason the interface to which the crypto map is applied is not used to route the traffic.

The concept of logical tunnel interfaces was developed, which would have a crypto map applied to

it. (This crypto map is not user configurable; however, the attributes that it uses are referenced from the user-configured IPsec profile used to protect the tunnel.) Any traffic sent or received on this logical interface would be protected by IPsec. This allows for traffic routing to be used to send traffic with the logical tunnel being the next hop and results in simplified configurations with greater flexibility for deployments. When the requirement for a user to configure what traffic was to be protected is removed, it reduces the chances of misconfigurations, which frequently occurred with manual configurations of access control lists with crypto maps.

On the Cisco Aggregation Services Routers (ASR), for every line or access control entry (ACE) in an access control list, a ternary content-addressable memory (TCAM) entry will be used. TCAM has a limited number of entries, and for crypto maps that contain a large number of ACEs, the TCAM can become exhausted. Tunnel protection uses only a single TCAM entry and allows for a larger number of IPsec Security Associations to be established compared to using crypto maps.

Crypto maps cannot natively support the use of MPLS, so the use of tunnel interfaces allows MPLS to be incorporated into a configuration.

Interface Types

When tunnels are used with IPsec, there are two types of tunnel interface encapsulation modes: pure IPsec or GRE.

Virtual Interfaces: VTI and GRE/IPsec

IPsec tunnels can be set up statically or dynamically using virtual interfaces of type VTI (Virtual-Tunnel Interface) or GRE/IPsec. These types of interfaces already existed in legacy IKEv1, and their use has been extended in FlexVPN.

The difference between the two interface types is the sequence of encapsulation: VTI (**tunnel mode IPsec {ipv4 | ipv6}**) carries IPv4 or IPv6 traffic directly within IPsec tunnel mode, while GRE/IPsec (**tunnel mode gre {ip | ipv6}**) first encapsulates traffic within GRE and a new IP header before encapsulating the resulting GRE/IP packet within IPsec transport mode.

For each type of virtual interface, two flavors are available: static (**interface tunnel** *number*) or dynamically instantiated (**interface virtual-access** *number*, dynamically created from a statically configured template-interface, **interface virtual-template** *number* **type tunnel**). You can think of virtual-template interface as a configuration template from which the virtual-access (VA) interfaces are dynamically cloned/instantiated.

In the rest of this section, we will simply refer to a “virtual interface” or “tunnel interface” when the described behavior remains the same regardless of the type (VTI or GRE/IPsec) and nature (static or dynamic) of the interface.

A virtual interface generally represents a point-to-point link that uses a specific protocol between two endpoints. The tunnel endpoints are configured using the **tunnel source** (local endpoint) and **tunnel destination** (remote endpoint) commands. Multipoint interfaces (mentioned at the end of this section) are the exception, for they do not have a remote endpoint configured and represent a single entry point into multiple point-to-point links.

Traffic Selection by Routing

With a virtual interface, outgoing traffic gets selected for encapsulation when it is routed through the interface, and incoming traffic from the VPN gets injected into the router after decapsulation with the tunnel as input interface. In order to achieve traffic selection by routing, the virtual interface must be a layer 3 interface with an IP address configured on it (called the tunnel address) and static or dynamic routes to the remote prefixes must be present in the routing table, with a next hop that is seen as directly connected through the virtual interface (that is, the next hop must be in the same subnet as the tunnel IP address).

The following example shows a simple static GRE tunnel configuration. The traffic flows from the local LAN (10.10.1.0/24) to the remote LAN (10.10.210.0/24). A static route to the remote LAN has been configured with the IP address of the tunnel interface on the remote GRE endpoint as the next hop.

Click here to view code image

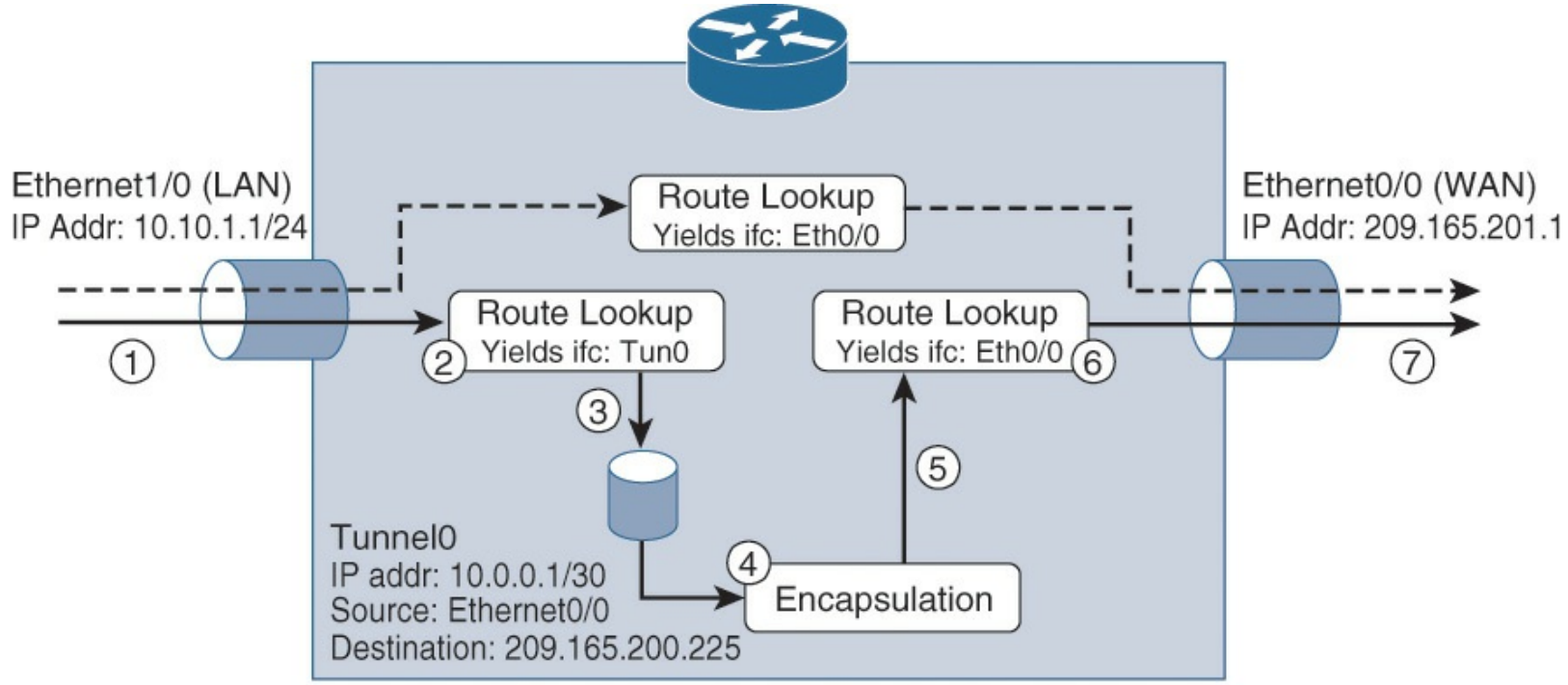
```
Router#show running-config interface Tunnel0
interface Tunnel0
 ip address 10.0.0.1 255.255.255.252
 tunnel source Ethernet0/0
 tunnel destination 209.165.200.225
end

Router#show running-config | include ip route
ip route 0.0.0.0 0.0.0.0 209.165.201.2
ip route 10.10.210.0 255.255.255.0 10.0.0.2

Router#show ip route 10.10.210.99
Routing entry for 10.10.210.0/24
  Known via "static", distance 1, metric 0
  Routing Descriptor Blocks:
  * 10.0.0.2
      Route metric is 0, traffic share count is 1

Router#show ip route 10.0.0.2
Routing entry for 10.0.0.0/30
  Known via "connected", distance 0, metric 0 (connected, via interface)
  Routing Descriptor Blocks:
  * directly connected, via tunnel0
      Route metric is 0; traffic share count is 1
```

Consider an inbound IP packet from the local LAN (source address 10.10.1.99) that is destined to the remote LAN (destination address 10.10.210.99). Figure 4-7 illustrates how this packet flows inside the router (plain line) as opposed to a packet destined to an address on the WAN (dotted line).



Routing Table (RIB)

	Destination	Next Hop and Interface
6	0.0.0.0/0	209.165.201.2 (Eth0/0)
	10.0.0.0/30	Connected (Tun0)
2	10.10.210.0/24	10.0.0.2 (Tun0)

Figure 4-7 *Packet Flow Though Tunnel Interface*

1. The packet enters through the LAN interface.
2. The route lookup for the destination (10.10.210.90, remote LAN) yields a next hop of 10.0.0.2, which is seen as directly connected on the tunnel interface (10.0.0.0/30).
3. The packet is handed over to the tunnel interface and the encapsulation code.
4. The packet is consumed during encapsulation (GRE/IPv4 in this case), and a new IP packet is created with the configured tunnel endpoints as its source and destination.
5. The new packet goes through the normal routing process.
6. The route lookup for the destination (209.165.200.225) yields the WAN ISP router as the next hop on the WAN interface.
7. The GRE packet is forwarded over the WAN interface.

Note that the source address is not taken into consideration when the traffic is selected, but only the destination address. This differs from what happens, for example, with IPsec encapsulation using a static crypto selector (crypto map with **match address** *access-list*).

Figure 4-8 shows the packet headers before and after encapsulation.

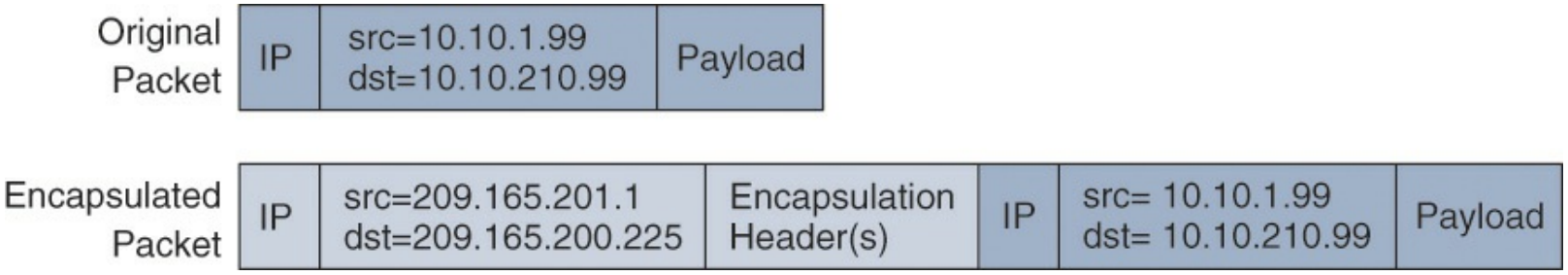


Figure 4-8 *Packet Headers and Encapsulation*

Static Tunnel Interfaces

Static virtual interfaces (IPsec tunnel or GRE/IPsec) can be used on initiators and responders; these interfaces remain in the configuration at all times and must specify the address of the peer VPN endpoint (**tunnel destination** *peer-address*). The following example illustrates a sample static VTI configuration.

Click here to view code image

```
interface Tunnel0
 ip address 10.0.0.1 255.255.255.252
 tunnel source Ethernet0/0
 tunnel mode ipsec ipv4
 tunnel destination 209.165.200.225
 tunnel protection ipsec profile default
```

When using the FlexVPN client profile feature (**crypto ikev2 client flexvpn**) (which will be described in chapter 10, "FlexVPN Client"), the peer address can optionally be set to **dynamic**. In this instance, the interface still remains in the configuration, and the peer address is populated at run time, based on the client profile configuration and tracking states.

Dynamic Tunnel Interfaces

Dynamic interfaces (IPsec tunnel or GRE/IPsec) are a responder-only feature, based on virtual-template interfaces that act as a template for dynamically created tunnels. A successful IKEv2 negotiation triggers the cloning of a virtual-template into a virtual-access interface (**interface Virtual-Access** *number*) that represents a point-to-point link to the remote peer, which is then protected by IPsec. The virtual-access interface remains up as long as the IPsec tunnel is up and gets deleted along with the corresponding IKE and IPsec Security Associations when the IPsec session is torn down.

> **Note**
>
> All virtual-template interfaces used in FlexVPN are declared as **type tunnel**, meaning that the instantiated virtual-access interfaces behave similar to static tunnel interfaces (**interface Tunnel** *number*): they perform some sort of encapsulation, can be configured with VRF (see later in this chapter), and generally support the same interface features.

Dynamic interfaces enable hub-and-spoke scenarios where the hub has a single virtual-template with the peer VPN endpoint left unspecified (no **tunnel destination** statement), while each spoke has a static tunnel interface that is configured with the hub IP address as the tunnel destination. In multi-

homed scenarios, where multiple IP addresses are on a VPN gateway, the local VPN endpoint (tunnel source statement) can also be left unspecified and be populated automatically based on the local address to which the peer is connecting to. The following example shows a sample Virtual-Template (dynamic VTI) configuration.

Click here to view code image

```
interface Loopback1
 ip address 10.0.0.1 255.255.255.255

interface Virtual-Template1 type tunnel
 ip unnumbered Loopback1
 tunnel mode ipsec ipv4
 tunnel protection ipsec profile default
```

While the virtual-template appears in the output of **show running-config**, the virtual-access interfaces do not. A specific command exists (**show derived-config**) that displays the configuration equivalent for the active virtual-access interfaces. The derived-config exists only at run time, is derived from the configuration on the virtual-template and from the IPsec session parameters, and is never written to the (non-volatile) NVRAM configuration, which would allow the configuration to be persistent even on a reload.

Note that the tunnel mode could be set to auto, and the resulting configuration applied on the cloned virtual-access interface would be based on what the peer proposes (tunnel mode gre/IPsec ipv4/ipv6).

The following example shows the derived config for a virtual-access interface based on the above virtual-template configuration. The highlighted lines have been automatically added during cloning.

Click here to view code image

```
router#show derived-config interface virtual-access 1
Building configuration...

Derived configuration : 225 bytes

interface Virtual-Access1
 ip unnumbered Loopback1
 tunnel source 209.165.200.225
 tunnel mode ipsec ipv4
 tunnel destination 209.165.201.1
 tunnel protection ipsec profile default
 no tunnel protection ipsec initiate
end
```

sVTI and dVTI

In the case of pure IPsec tunnel encapsulation (without GRE), one refers to a statically configured tunnel interface as an sVTI (static VTI) and to a virtual-template as a dVTI (dynamic VTI). A single site-to-site tunnel is typically sVTI-to-sVTI, while a hub-and-spoke setup is typically sVTI-to-dVTI, with the dVTI on the hub. In addition, a feature called multi-SA dVTI provides support for interoperating with non-VTI-compatible peers (for example, those using legacy crypto maps), where the dVTI will present a mirror image of the traffic selectors requested by the initiator and can terminate multiple IPsec Security Associations from the same peer on the same virtual-access interface.

Multipoint GRE

A particular type of GRE encapsulation is multipoint GRE (mGRE, **tunnel mode gre multipoint**) where a single static GRE tunnel interface is used as the endpoint for multiple site-to-site tunnels. DMVPN is based on this mode and uses a single interface on each hub as well as on each spoke to terminate all static and dynamic tunnels.

FlexVPN does not rely on multipoint interfaces.

The following example shows a sample spoke-side multipoint GRE/IPsec interface. The mGRE-specific configuration statements are highlighted.

Click here to view code image

```
interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 ip nhrp network-id 1
 ip nhrp nhs 10.0.0.251 nmba 209.165.200.225 multicast
 tunnel source Ethernet0/0
 tunnel mode gre multipoint
 tunnel protection ipsec profile default
```

An mGRE interface is essentially a GRE tunnel interface that has lost its point-to-point nature and acts as if it is directly connected to a cloud of remote mGRE interfaces. The tunnel IP addresses of all mGRE interfaces in that cloud are part of a single subnet, and their IP addresses are all seen as directly connected, as they would be (for example) on an Ethernet link.

This is made possible by the use of the Next Hop Resolution Protocol (NHRP), which acts as a resolution mechanism between a peer's tunnel address (the IP address configured on the peer's mGRE interface) and the mGRE endpoint address on that peer, called the Non-Broadcast Multiple Access (NBMA) address.

The term "non-broadcast" expresses the fact that it is not possible to simply send a request containing the tunnel address being resolved to all possible peers and expect the owner of that address to reply, as the mGRE endpoints are not necessarily part of the same broadcast domain: they may, for example, be reachable through the Internet, dispersed across a large corporate network, or reachable over different interfaces and media.

Consider a packet flowing from the local LAN to the LAN of one of the peers on the mGRE interface in the example above. Figure 4-9 shows the different steps in performing the mGRE encapsulation. In this simplified example, we assume that the necessary NHRP mappings are already populated in the NHRP cache and that a routing protocol is used to advertise prefixes reachable over mGRE.

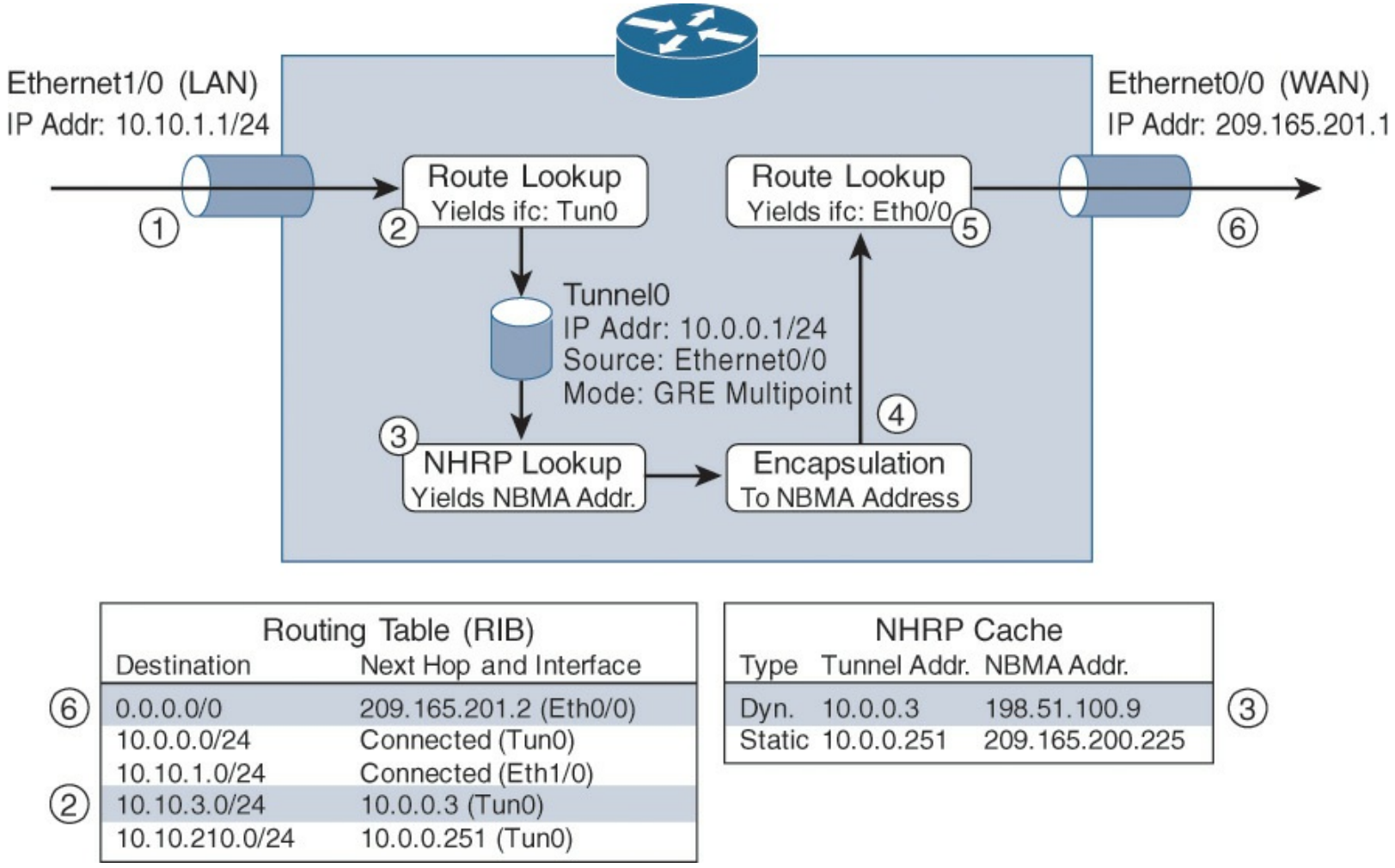


Figure 4-9 *Encapsulation When Using mGRE*

1. The packet enters through the LAN interface with source 10.10.1.99 and destination 10.10.3.99.
2. The route lookup for the destination (10.10.3.99) yields a next hop of 10.0.0.3, which is seen as directly connected on the tunnel interface (10.0.0.0/24).
3. The multipoint GRE code performs a lookup in the NHRP cache to resolve the remote tunnel address (10.0.0.3) into the corresponding NBMA address (198.51.100.9).
4. The original packet is consumed during encapsulation, and a new GRE/IP packet is created with the WAN interface address as its source and the remote NBMA address as its destination.
5. The route lookup for the destination (198.51.100.9) yields the WAN ISP router as the next hop on the WAN interface.
6. The GRE packet is forwarded over the WAN interface.

Note that Figure 4-9 describes mGRE encapsulation from a purely functional standpoint; in practice, Cisco Express Forwarding (CEF) accelerates steps 2 to 5 by setting up adjacencies on the tunnel interface based on the entries in the NHRP cache (to avoid a table lookup for every packet) and by prepopulating the Forwarding Information Base (FIB) with a complete output chain for the remote destinations.

Tunnel Protection and Crypto Sockets

All virtual-interface types mentioned so far perform some form of IPsec encapsulation. This is achieved by configuring the **tunnel protection** statement on the VTI or GRE/IPsec tunnel interface. This command applies a construct called IPsec profile to the interface and ensures that the crypto layer will be hit by packets traversing the interface (immediately in the case of VTI, and right after GRE encapsulation in the case of GRE/IPsec).

The following example shows a GRE/IPsec tunnel configuration (since the default tunnel mode is **tunnel mode gre ip**, the command does not appear in the running configuration). The default IPsec profile is applied and links to the default IKEv2 profile.

Click here to view code image

```
crypto ipsec profile default
 set ikev2-profile default

interface Tunnel0
 ip address 10.0.0.1 255.255.255.252
 tunnel source Ethernet0/0
 tunnel destination 209.165.200.225
 tunnel protection ipsec profile default
```

The tunnel protection feature relies on a mechanism called crypto sockets that represents crypto sessions in a way similar to network sockets: there are listening sockets and established sockets, which are dynamically created and deleted on demand. Every IPsec VPN session (bundle of IKE and IPsec Security Associations with a specific peer) results in a separate crypto socket.

The following example shows sample output of **show crypto sockets** for the interface configuration in the previous example. The IPsec tunnel is established (one active socket), and IPsec is listening for incoming sessions (one listening socket).

Click here to view code image

```
router#show crypto sockets

Number of Crypto Socket connections 1

   Tu0 Peers (local/remote): 209.165.201.1/209.165.200.225
       Local Ident  (addr/mask/port/prot): (209.165.201.1/255.255.255.255/0/47)
       Remote Ident (addr/mask/port/prot): (209.165.200.225/255.255.255.255/0/47)
       IPSec Profile: "default"
       Socket State: Open
       Client: "TUNNEL SEC" (Client State: Active)

Crypto Sockets in Listen state:
Client: "TUNNEL SEC" Profile: "default" Map-name: "Tunnel0-head-0"
```

Another convenient command when working with IPsec VPNs on IOS is **show crypto session**. It displays the active IKEv1 and IKEv2 security associations (SAs) as well as the active IPsec Security Associations. The following example illustrates this command applied to the example above.

Click here to view code image

```
router#show crypto session
Crypto session current status

Interface: Tunnel0
```

```
Session status: UP-ACTIVE
Peer: 209.165.200.225 port 500
  IKEv2 SA: local 209.165.201.1/500 remote 209.165.200.225/500 Active
  IPSEC FLOW: permit 47 host 209.165.201.1 host 209.165.200.225
        Active SAs: 2, origin: crypto-map
```

While those two commands may look similar at first sight, they actually represent two different functions. Crypto sockets are a representation of the IPsec state and are independent of the IKE layer. They are used by the tunnel protection mechanism; however, a legacy crypto map configuration would not rely on them, and **show crypto sockets** for a crypto map tunnel would not return any output. On the other hand, **show crypto session** offers a summarized view of the IKE and IPsec Security Associations in the system for each peer, regardless of the origin of those SAs, whether crypto map or tunnel protection.

Implementation Modes

This section describes the ability of GRE and native IPsec tunnel encapsulation modes to carry IPv4 and/or IPv6 traffic and the ability to automatically detect the tunnel encapsulation mode and IP transport type on an IKEv2 responder.

Dual Stack

It is now common for devices to support both IPv4 and IPv6 protocols at the same time. Many operating systems now ship with both protocols enabled by default, and networks that are looking to transition from IPv4 to IPv6 commonly require both protocols to coexist, allowing devices to access both IPv4 and IPv6 content in parallel.

As you learned from chapter 2, "IKEv2: The Protocol", multiple traffic selectors can be created within the IKEv2 CREATE_CHILD_SA exchange, thus allowing a single IPsec Security Association to transport protocols of different types. This can allow both IPv4 and IPv6 traffic to be created using a single IPsec Security Association. The ability to tunnel both IPv4 and IPv6 traffic using a single SA over IPv4 or IPv6 transport is referred as dual stack support.

Cisco IOS allows for dual stack support on IKEv2 gateways when using the Cisco AnyConnect client; however the transport protocol used must be IPv4, and the encapsulation must be GRE. However, both IPv6 and IPv4 addresses can be dynamically obtained over this, and data can pass over the relevant traffic selectors. Dual stack is achieved when the AnyConnect client advertises that it supports this ability by sending a Vendor ID payload of type Cisco AnyConnect GRE Mode. The IKEv2 headend will then allocate the AnyConnect client an IPv4 or IPv6 (or both if required) address via Configuration Payload. If the AnyConnect client requests both an IPv4 and an IPv6 address, then traffic selectors for both IPv4 and IPv6 traffic will be requested, and the headend will install a single traffic selector for IPv4 using GRE between both hosts. All traffic will run over this SA.

It should be noted that Cisco IOS naturally allows for the transportation of both IPv4 and IPv6 traffic when GRE encapsulation is used with IPsec. Because GRE is used as the encapsulation mechanism, it can inherently encapsulate either protocol and will only require a single IPsec Security Association that contains the tunnel source and destination IP addresses and the protocol, which is GRE (47).

Mixed Mode

On Cisco IOS, virtual interfaces implementing IPsec tunnel mode with IPv4 (**tunnel mode IPsec ipv4**) or IPv6 (**tunnel mode IPsec ipv6**) transport will by default negotiate IPsec Security Associations that protect the same protocol version as the one used for transport (for instance, an IPsec virtual interface using IPsec over IPv4 will negotiate protection only for IPv4 traffic using IP protocol 4 (IPv4 0×04), whereas IPv6 uses IP protocol 41 (0×29).

Since 15.6(1)T, Cisco IOS can use a tunnel interface configured in IPsec tunnel mode with IPv4 to create IPv6 Security Associations, or IPsec tunnel mode with IPv6 to create IPv4 Security Associations by using the **tunnel mode IPsec ipv4 v6-overlay** and **tunnel mode IPsec ipv6 v4-overlay** commands respectively. This allows an IPv4 overlay network to be extended over an IPv6 transport network, or an IPv6 overlay network to be extended over an IPv4 transport network when IPsec tunnel mode virtual interfaces are used.

The following example illustrates the configuration used for the creation of an IPv4 tunnel over an IPv6 transport using a VTI.

Click here to view code image

```
interface Tunnel0
    ip address 192.168.1.1 255.255.255.0
    tunnel source GigabitEthernet0/0
    tunnel mode ipsec ipv6 v4-overlay
    tunnel destination 2001:DB8::2
    tunnel protection ipsec profile default
```

The following illustrates the established session. The IKEv2 SA is created between the IPv6 addresses, and note that the created IPsec Security Associations are using IPv4 traffic selectors (0.0.0.0/0.0.0.0).

Click here to view code image

```
Router#show crypto session
Crypto session current status
Interface: Tunnel0
Profile: default
Session status: UP-ACTIVE
Peer: 2001:DB8::2 port 4500
  Session ID: 1
  IKEv2 SA: local 2001:DB8::1/500 remote 2001:DB8::2/500 Active
  IPSEC FLOW: permit ip 0.0.0.0/0.0.0.0 0.0.0.0/0.0.0.0
        Active SAs: 2, origin: crypto-map
```

Figure 4-10 illustrates an IPv4 packet being encapsulated within an IPv6 tunnel when IPsec tunnel mode is used.

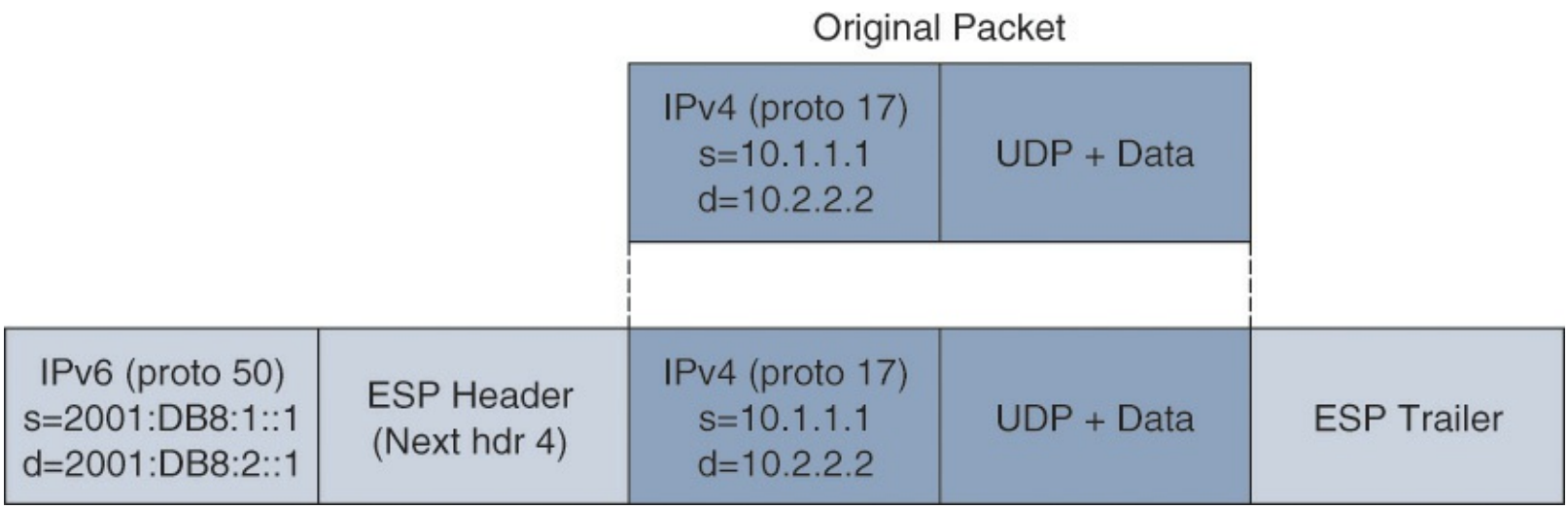


Figure 4-10 *Mixed-Mode v4 over v6 Tunnel*

To configure an IPv6 overlay over IPv4 tunnel, the following tunnel mode must be used:

Click here to view code image

```
tunnel mode ipsec ipv4 v6-overlay
```

The following example illustrates the configuration used for the creation of an IPv6 tunnel over an IPv4 transport that uses a VTI.

Click here to view code image

```
interface Tunnel0
    ipv6 address 2001:DB8::1/112
    ipv6 enable
    tunnel source GigabitEthernet0/0
    tunnel mode ipsec ipv4 v6-overlay
    tunnel destination 192.168.1.2
    tunnel protection ipsec profile default
```

The following illustrates the established session. The IKEv2 SA is created between the IPv4 addresses, and note that the created IPsec Security Associations are using IPv6 traffic selectors (::/0).

Click here to view code image

```
Router#show crypto session
Crypto session current status
Interface: Tunnel0
Profile: default
Session status: UP-ACTIVE
Peer: 192.168.1.2 port 4500
  Session ID: 1
  IKEv2 SA: local 192.168.1.1/500 remote 192.168.1.2/500 Active
  IPSEC FLOW: permit ipv6 ::/0 ::/0
        Active SAs: 2, origin: crypto-map
```

Figure 4-11 illustrates an IPv6 packet that is encapsulated within an IPv4 tunnel when IPsec tunnel mode is used.

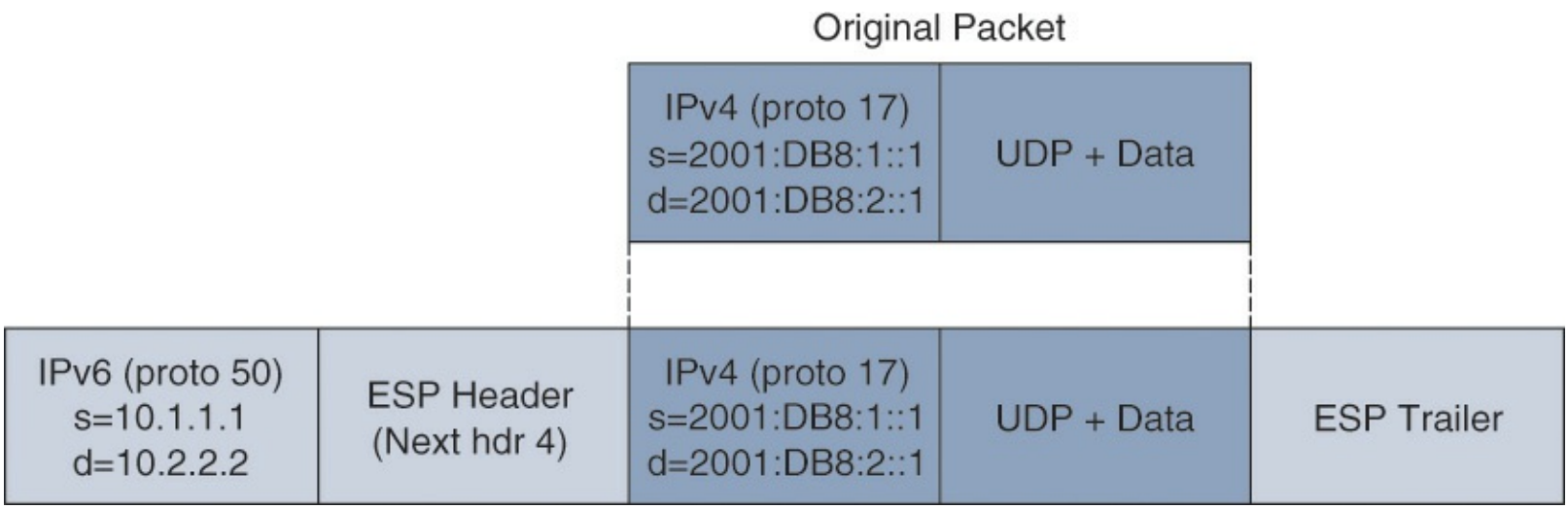


Figure 4-11 *Mixed-Mode v6 over v4 Tunnel*

Since 15.2(3)T, Cisco IOS (when acting as remote access VPN headend) can support sending IPv4 traffic over an IPv6 Security Association when IPsec tunnel mode is used as the encapsulation protocol (**tunnel mode IPsec IPv6**), or IPv6 traffic over an IPv4 Security Association when IPsec tunnel mode is used as the encapsulation protocol (**tunnel mode IPsec IPv4**).

There are a number of caveats to using this behavior: the IOS device must be using IKEv2, and the tunnel interface must be configured as a dynamic virtual-tunnel interface.

To enable activation of mixed mode, the **set mixed-mode** command must be enabled under the IPsec profile.

Note that when using a tunnel interface with GRE encapsulation, the limitation of requiring a certain version of IOS to transport IPv4 traffic within a IPv6 tunnel or IPv6 traffic within an IPv4 tunnel does not occur. When GRE encapsulation is used with tunnel protection, a single IPsec Security Association will be created that can then transport both IPv4 and IPv6 traffic within the encapsulated tunnel. This is achieved by using GRE/IPsec, where IPv4 and IPv6 are both first encapsulated into GRE and then into either IPv4 or IPv6 using IPsec transport mode. GRE acts as an abstraction layer that isolates the inner protocol from the IPsec layer, which handles only the protection of GRE packets over either IPv4 or IPv6 (depending on the tunnel endpoints).

Auto Tunnel Mode

Cisco IOS and IOS-XE (since 15.4(2)T) allows a router acting as a IKEv2 responder to use Auto Tunnel Mode (the **virtual-template** being configured with the **mode auto** command). In this mode, the headend will mimic the mode presented by the peer and respond by creating an IPsec Security Association that uses the same attributes (the choice being GRE or IPsec with IPv4 or IPv6). This allows a headend to serve multiple clients of different tunnel types that use both IPv4 and IPv6 with minimal configuration.

VRF-Aware IPsec

This section describes VRF as it applies to IPsec.

VRF in Brief

Virtual Routing and Forwarding (VRF) technology makes it possible to virtualize layer 3 forwarding on a router and implement multiple separate routing domains on a single device. It enables segregation of traffic flows from different parts of the network or from different customers (known as multitenant) as well as overlapping addresses. A VRF-aware router will maintain separate routing instances with separate information per VRF, including a separate Routing Information Base (RIB), Forwarding Information Base (FIB), ARP table, and routing protocol information. The default VRF on a router is global. VRF can be encountered in two forms: MPLS VPN and VRF-Lite.

A VRF needs to be defined and given a name by using the commands **ip vrf** (deprecated, IPv4 only) or **vrf definition** (multiprotocol, supporting the IPv4 and IPv6 address-families). Each physical and logical (for example, loopback or tunnel) interface on the device can be assigned to a single VRF by using the commands **ip vrf forwarding** (deprecated, for IPv4-only VRF) or **vrf forwarding** (multiprotocol supporting IPv4 and IPv6). By default, every interface belongs to the global VRF. Traffic coming in through an interface is forwarded according to the routing table of the VRF configured on the interface.

Without specific configuration, all traffic will leave the router through the same VRF as it entered. Packets can enter the router through one VRF and leave through another in a number of situations:

- encapsulation/decapsulation (described later in this section)
- route leaking, where routes are set up statically or imported via BGP, with the next hop pointing to an interface in another VRF
- route replication, which is used with Easy Virtual Network (EVN), where prefixes are copied directly between VRFs using the Routing Information Base (RIB)

The following example shows the definition of two VRFs named red and blue. Figure 4-12 illustrate the independent forwarding of traffic within the two VRFs. The overlap between the addresses on Ethernet0/1 and Ethernet1/1 has no effect, because they belong to separate routing domains.

Click here to view code image

```
vrf definition blue
 address-family ipv4
 exit-address-family

vrf definition red
 address-family ipv4
 exit-address-family

interface Ethernet0/1
 vrf forwarding red
 ip address 172.16.10.1 255.255.255.0

interface Ethernet0/2
 vrf forwarding red
 ip address 172.16.20.1 255.255.255.0

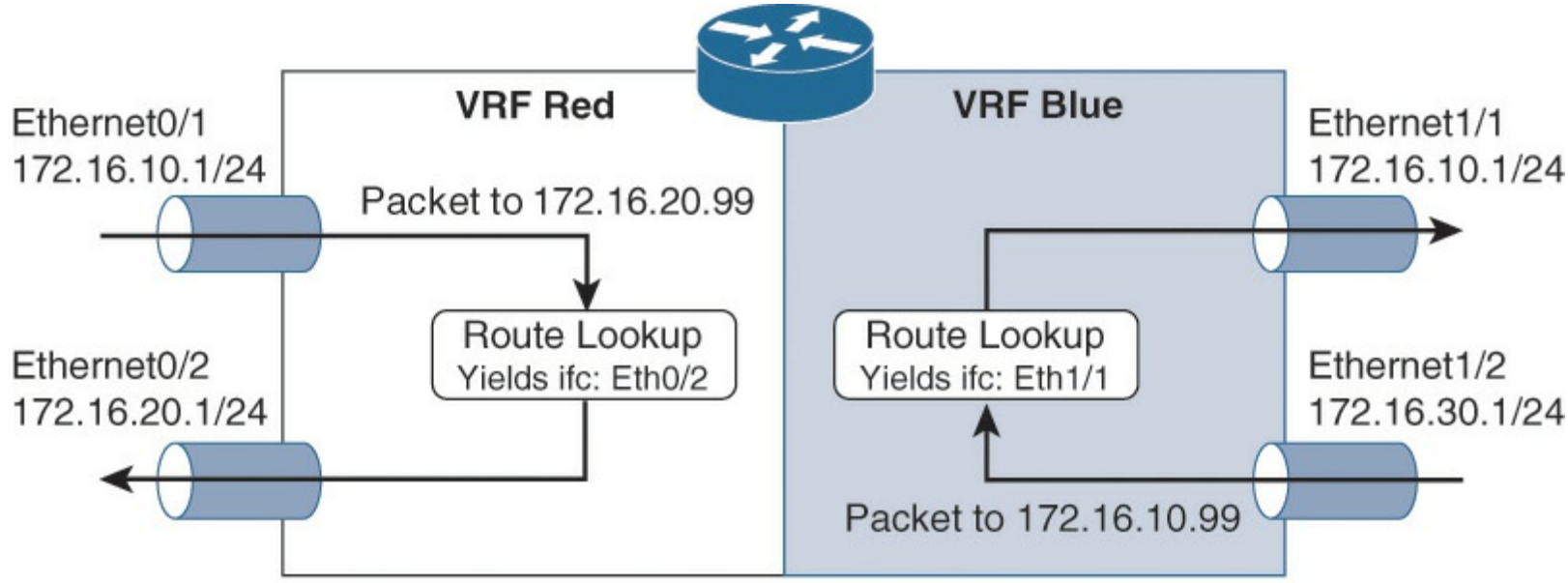
interface Ethernet1/1
 vrf forwarding blue
 ip address 172.16.10.1 255.255.255.0

interface Ethernet1/2
 vrf forwarding blue
```

```
 ip address 172.16.30.1 255.255.255.0

s1#show ip route vrf red
Routing Table: red
[...]
      172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks
C        172.16.10.0/24 is directly connected, Ethernet0/1
L        172.16.10.1/32 is directly connected, Ethernet0/1
C        172.16.20.0/24 is directly connected, Ethernet0/2
L        172.16.20.1/32 is directly connected, Ethernet0/2

s1#show ip route vrf blue
Routing Table: blue
      172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks
C        172.16.10.0/24 is directly connected, Ethernet1/1
L        172.16.10.1/32 is directly connected, Ethernet1/1
C        172.16.30.0/24 is directly connected, Ethernet1/2
L        172.16.30.1/32 is directly connected, Ethernet1/2
```



VRF **Red** Routing Table (RIB)

Destination	Next Hop and Interface
172.16.10.0/24	Connected (Eth0/1)
172.16.20.0/24	Connected (Eth0/2)

VRF **Blue** Routing Table (RIB)

Destination	Next Hop and Interface
172.16.10.0/24	Connected (Eth1/1)
172.16.30.0/24	Connected (Eth1/2)

Figure 4-12 *Forwarding When Using VRFs*

Features that offer support for VRF are called VRF aware. This can mean different things for different features; for instance, a VRF-aware SNMP MIB will provide separate results per VRF and may restrict access from inside a certain VRF to OIDs pertaining to other VRFs, or that if a feature is not VRF aware it cannot be used when accessed from a VRF aware interface.

VRF-Aware GRE and VRF-Aware IPsec

GRE is VRF aware, in the sense that the encapsulated (payload) and nonencapsulated (delivery) traffic may reside in different VRFs. This is achieved by proper configuration of the GRE tunnel interface (see below).

IPsec is also VRF aware, both with legacy crypto maps and VTI. The configuration differs slightly: legacy VRF awareness (not covered in this book) is based on IKE profiles and route leaking, while

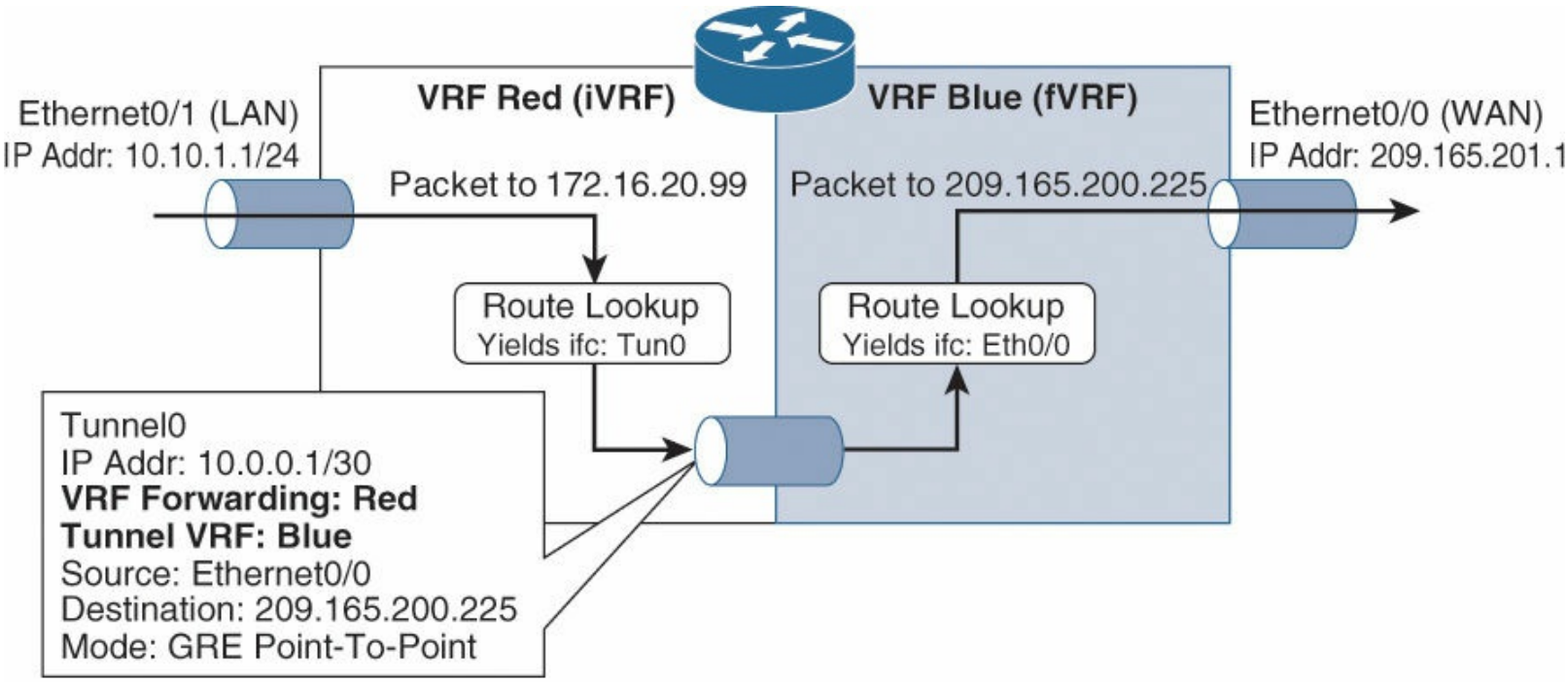
VTI only relies on the VRF configuration on the tunnel interface being the same as GRE. In general, encapsulation through a tunnel interface is VRF aware.

Throughout this book, we will refer to the VRF on which the nonencapsulated (pre-encaps or post-decaps) traffic resides as the IVRF or inside VRF, and to the VRF where encapsulated traffic resides as the FVRF or front-door VRF. The IP address configured on the tunnel interface (the tunnel address) resides in the IVRF, while the tunnel source and destination reside in the FVRF.

The IVRF of a tunnel interface is configured with the **vrf forwarding *name*** command. If the interface is **{ip | ipv6} unnumbered** to another interface, the two interfaces must reside in the same VRF (with the same **vrf forwarding** statements).

The FVRF on a tunnel is configured with the **tunnel vrf *name*** command.

Figure 4-13 illustrates VRF-aware GRE encapsulation.



VRF **Red** Routing Table (RIB)

Destination	Next Hop and Interface
10.0.0.0/30	Connected (Tun0)
10.10.1.0/24	Connected (Eth1/0)
10.10.210.0/24	10.0.0.2 (Tun0)

VRF **Blue** Routing Table (RIB)

Destination	Next Hop and Interface
0.0.0.0/0	209.165.201.2 (Eth0/0)

Figure 4-13 *VRF-Aware GRE Encapsulation*

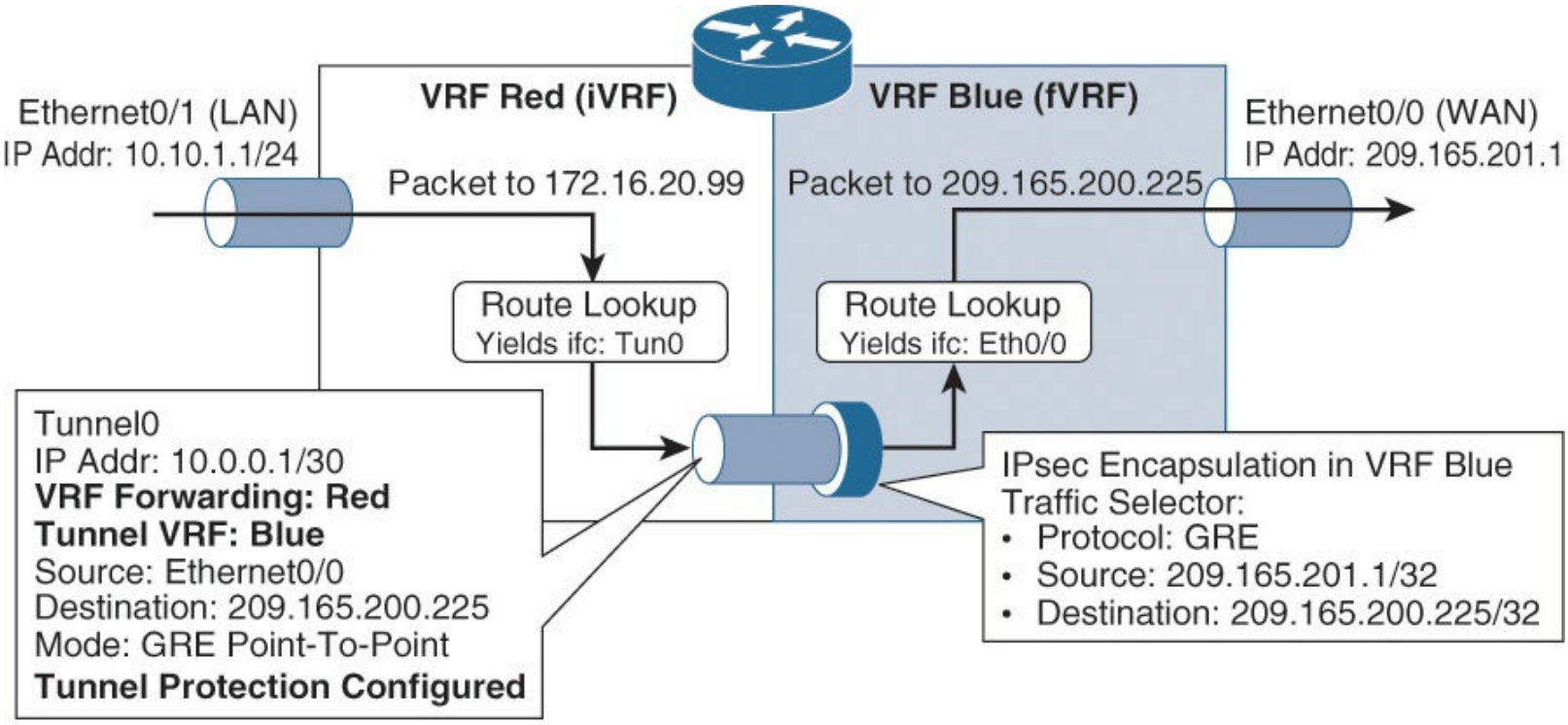
VRF-Aware GRE over IPsec

GRE over IPsec is configured by adding a **tunnel protection** statement on a GRE tunnel interface (**tunnel mode gre {ip | ipv6}**). Even though the resulting packet contains both an IPsec and a GRE header, the two encapsulation steps do not occur at exactly the same stage: the tunnel interface still represents a tunnel that performs GRE encapsulation only, and the IPsec feature is set up to protect the resulting GRE packet.

In terms of VRF awareness, this means that the hop between IVRF and FVRF is performed by the sole

GRE encapsulation, while the GRE/IP traffic is selected, encrypted, and decrypted by IPsec within the FVRF.

Figure 4-14 illustrates the VRF-aware GRE/IPsec packet flow; the only difference with Figure 4-13 is the addition of the IPsec post-encapsulation feature on the GRE tunnel interface.



VRF **Red** Routing Table (RIB)

Destination	Next Hop and Interface
10.0.0.0/30	Connected (Tun0)
10.10.1.0/24	Connected (Eth1/0)
10.10.210.0/24	10.0.0.2 (Tun0)

VRF **Blue** Routing Table (RIB)

Destination	Next Hop and Interface
0.0.0.0/0	209.165.201.2 (Eth0/0)

Figure 4-14 *VRF-Aware GRE Packet Flow*

Summary

By now you should be familiar with the modes of IPsec and the overhead that is generated when using transport mode or tunnel mode. You will also know that Generic Routing Encapsulation (GRE) is used for encapsulating non-IP protocols and what overhead using GRE incurs.

This chapter detailed the various implementation modes on Cisco IOS and explained how you can use mixed mode to transport IPv4 over IPv6 tunnel interfaces, or IPv6 over IPv4 tunnel interfaces. We also demonstrated how to use auto tunnel mode, whereby the tunnel mode and tunnel protection is automatically replicated from the incoming traffic selectors, allowing for a single tunnel interface to be used by peers with different tunnel types.

Within this chapter we covered the building blocks that show how tunnel interfaces operate on Cisco IOS, and we described what tunnel modes you can configure for IPsec and the benefits that each brings. Virtual Tunnel Interfaces are used when GRE is not required. Static VTIs are used when the peer is known, while Dynamic VTIs are a responder-only feature and allow a virtual-access interface to be spawned from the virtual-template.

The method of route lookup and the ability for traffic to move between VRF's when traversing a

tunnel were described, along with the security benefits that this brings.

Reference

http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/evn/configuration/xe-3s/evn-xe-3s-book/evn-shared-svcs.html

Part IV: IKEv2 Implementation

Chapter 5. IKEv2 Configuration

This chapter introduces IKEv2 configuration on Cisco IOS devices. The chapter assumes that the reader is familiar with the basics of IPsec VPNs, the IKEv2 protocol, and Cisco IOS IPsec implementation covered in the previous chapters. The chapter covers the foundational IKEv2 configuration topics in detail, with a focus on the rationale behind configuration and how it relates to the IKEv2 protocol. The chapter also covers smart defaults and the relevant PKI and IPsec configurations required to set up point-to-point IKEv2-based IPsec VPN tunnels between Cisco IOS devices.

The IKEv2 configuration topics that are specific to FlexVPN Client, FlexVPN Server, and IKEv2 Server Load Balancing are covered in the respective chapters.

The chapter covers the following main topics

IKEv2 Configuration Overview

IKEv2 Proposal

IKEv2 Policy

IKEv2 Keyring

IKEv2 Profile

IKEv2 Global Configuration

PKI Configuration

IPsec Configuration

Smart Defaults

IKEv2 Configuration Overview

This section provides a high-level overview of the IKEv2 configuration, including the guiding principle, the scope, and the purpose of the IKEv2 configuration constructs.

The Guiding Principle

One of the goals of the Cisco IKEv2 configuration is to keep the configuration intuitive and minimal. The user needs to configure only the mandatory parameters, while smart defaults are used wherever possible. Smart defaults use default values that cater to most of the use cases and can be customized to meet the needs of specific users.

Scope of IKEv2 Configuration

IKEv2 configurations have a well-defined scope, which is either context-specific or global. The context-specific configuration is used for a specific group of peers and/or negotiation context, such as VRF and local address, and this configuration is either defined in or is referenced from an IKEv2 profile or IKEv2 policy, both described later in the chapter. The global configuration is used irrespective of the context (that is, for all peers and negotiations) and is defined globally in the router configuration mode. For configuration that is defined both globally and in a specific context, such as under IKEv2 profile, the latter takes precedence.

IKEv2 Configuration Constructs

Table 5-1 lists the IKEv2 configuration constructs and shows if the configuration is mandatory, if it has a smart default, and when it must be configured.

IKEv2 Configuration	Mandatory	Smart Default	When to Configure
IKEv2 Profile	Yes	No	Always configure IKEv2 profile, as it defines the authentication method and credentials required for the negotiation
IKEv2 Proposal	No	Yes	Configure IKEv2 proposal if the default proposal does not meet the requirement
IKEv2 Policy	No	Yes	Configure IKEv2 policy if the default policy does not meet the requirement
IKEv2 Keyring	No	No	Configure IKEv2 keyring when pre-shared key authentication is used
IKEv2 Global Configuration	No	No	Configure global IKEv2 parameters to apply to all peers

Table 5-1 *IKEv2 Configuration Constructs*

Note that except for the IKEv2 global configuration, the scope of IKEv2 configuration is defined either by the IKEv2 profile or IKEv2 policy, as these configurations must be referenced from the IKEv2 profile or IKEv2 policy. Also, note that while the IKEv2 policy defines the scope of the IKEv2 proposals, the IKEv2 profile defines the scope of rest of the configurations.

IKEv2 Proposal

The IKEv2 proposal defines cryptographic transforms that are negotiated in the IKE_SA_INIT exchange and are used to protect the IKEv2 Security Association that is to be created. The transform types configured are encryption algorithm, pseudorandom function, integrity algorithm, and Diffie-Hellman group. Note that unlike IKEv1, IKEv2 does not negotiate authentication method and SA lifetime, so the proposal does not include them.

The IKEv2 proposal mimics and corresponds to the proposal substructure in the IKEv2 Security Association (SA) payload where an SA payload can have one or more proposals and each proposal can have one or more transforms of each type.

Any number of IKEv2 proposals can be configured on a device. If a device is to support peers with different sets of cryptographic algorithms, then a single IKEv2 proposal containing all the algorithms that are to be supported can be configured. However if this causes an issue for whatever reason, then multiple proposals can be configured, with each being selected based on criteria set in the IKEv2 policy.

One scenario where multiple IKEv2 proposals must be configured is when an initiator wants to propose both combined-mode ciphers (for example, AES-GCM) and normal ciphers (for example, AES-CBC). In this case, two separate proposals must be defined, one with combined-mode ciphers and the other with normal ciphers with the integrity algorithms. Note that unlike normal ciphers,

combined-mode ciphers include both integrity and encryption in a single encryption algorithm and do not use an integrity algorithm. The IKEv2 RFC requires that combined-mode and normal ciphers be carried in separate proposals in the SA payload.

Another reason to define multiple IKEv2 proposals occurs when only specific combinations of transform types are required, as a proposal with multiple transforms of each type makes all possible combinations available to the responder to select from.

The IKEv2 proposal is called from within the IKEv2 policy, which will match on certain criteria (FVRF and/or local IP address). Once the IKEv2 proposal has been configured, it should be referenced within the IKEv2 policy to ensure it is activated when needed.

The following rules apply to the use of IKEv2 proposals

- At least one transform for each type must be configured without which an IKEv2 proposal is considered incomplete and is not used in the negotiation. The only exception is when a combined-mode cipher is configured in IKEv2 proposal where an integrity algorithm is not required.
- When multiple transforms of each type are configured, they must be listed in the order of most to least preferred from left to right.
- When the IKEv2 initiator and responder have proposals with conflicting preferences, the initiator's preference is honored.
- An IKEv2 proposal must be referenced from an IKEv2 policy that defines the scope or the context in which the proposal is used.
- For the IKE SA negotiation to be successful, the responder must have at least one IKEv2 proposal matching the transforms proposed by initiator.
- Combined-mode ciphers and normal ciphers cannot be combined in a single IKEv2 proposal.

Configuring the IKEv2 Proposal

The IKEv2 proposal is created using the following command in the router global configuration mode

Click here to view code image

```
crypto ikev2 proposal proposal-name
```

Once the proposal is created, the router command prompt will change from **(config)** to **(config-ikev2-proposal)** denoting that user is in IKEv2-proposal sub-mode, and a message is displayed describing the requirements for a proposal to be complete. The following outputs shows creating an IKEv2 proposal, commands configurable under IKEv2 proposal, displaying the configured proposals, referencing the proposal from IKEv2 policy, and deleting the proposal.

Click here to view code image

```
Router(config)#crypto ikev2 proposal ikev2_proposal
IKEv2 proposal MUST either have a set of an encryption algorithm other than
aes-gcm, an integrity algorithm and a DH group configured or
 encryption algorithm aes-gcm, a prf algorithm and a DH group configured
Router(config-ikev2-proposal)#
```

Under the IKEv2-proposal sub-mode, the user can configure one or more encryption algorithms, integrity algorithms, pseudorandom functions (PRFs), and Diffie-Hellman groups.

Click here to view code image

```
Router(config-ikev2-proposal)#?
IKEv2 Proposal commands:
  encryption  Set encryption algorithm(s) for proposal
  exit        Exit from IKEv2 proposal configuration mode
  group       Set the Diffie-Hellman group(s)
  integrity   Set integrity hash algorithm(s) for proposal
  no          Negate a command or set its defaults
  prf         Set prf algorithm(s) for proposal
```

If an IKEv2 proposal does not contain all types of algorithms then this will not be used, this will be noted by the **Proposal Incomplete** message, which is displayed in the running configuration as shown:

Click here to view code image

```
Router#show run | begin ikev2 proposal
crypto ikev2 proposal ikev2_proposal
 ! Proposal Incomplete(MUST have atleast an encryption algorithm, an integrity
algorithm and a dh group configured)
```

The configured IKEv2 proposals can be shown using the following command:

Click here to view code image

```
Router#show crypto ikev2 proposal
 IKEv2 proposal: default
     Encryption : AES-CBC-256 AES-CBC-192 AES-CBC-128
     Integrity  : SHA512 SHA384 SHA256 SHA96 MD596
     PRF        : SHA512 SHA384 SHA256 SHA1 MD5
     DH Group   : DH_GROUP_1536_MODP/Group 5 DH_GROUP_1024_MODP/Group 2
 IKEv2 proposal: ikev2_proposal
     Encryption : AES-GCM-256 AES-GCM-128
     Integrity  : none
     PRF        : SHA256 SHA384
     DH Group   : DH_GROUP_384_ECP/Group 20 DH_GROUP_521_ECP/Group 21
```

The proposal must be referenced from IKEv2 policy in order to be used during negotiation.

Click here to view code image

```
Router(config)#crypto ikev2 policy ikev2_policy
Router(config-ikev2-policy)#proposal ikev2_proposal
```

The proposal must be dereferenced from IKEv2 policy before the proposal can be deleted.

Click here to view code image

```
Router(config)#no crypto ikev2 proposal ikev2_proposal
% Cannot remove as proposal is in use.
Router(config)#crypto ikev2 policy ikev2_policy
Router(config-ikev2-policy)#no proposal ikev2_proposal
Router(config-ikev2-policy)#exit
Router(config)#no crypto ikev2 proposal ikev2_proposal
Router(config)#
```

The following output shows the creation of an IKEv2 proposal with normal ciphers (for example, AES-CBC), along with the necessary integrity algorithm, pseudorandom function, and Diffie-Hellman groups configured. Note that when a normal cipher is configured, the combined-mode cipher option is not available.

Click here to view code image

```
Router(config)#crypto ikev2 proposal ikev2_proposal
Router(config-ikev2-proposal)#encryption aes-cbc-256 ?
  3des         3DES
  aes-cbc-128  AES-CBC-128
  aes-cbc-192  AES-CBC-192
  des          DES
  <cr>

Router(config-ikev2-proposal)#encryption aes-cbc-256
Router(config-ikev2-proposal)#integrity sha512
Router(config-ikev2-proposal)#prf sha512
Router(config-ikev2-proposal)#group 16
```

The configured proposal can be seen in running configuration and in the show command output:

Click here to view code image

```
Router#show run | begin ikev2 proposal
crypto ikev2 proposal ikev2_proposal
 encryption aes-cbc-256
 integrity sha512
 group 16

Router#show crypto ikev2 proposal ikev2_proposal
 IKEv2 proposal: ikev2_proposal
     Encryption : AES-CBC-256
     Integrity  : SHA512
     PRF        : SHA512
     DH Group   : DH_GROUP_4096_MODP/Group 16
```

The following example shows the creation of an IKEv2 proposal with combined-mode ciphers (AES-GCM), along with the necessary pseudorandom function and Diffie-Hellman groups configured. Note that when a combined-mode cipher is configured, the normal-mode cipher option is not available, configuration of an integrity algorithm is not allowed, and the proposal is marked as complete without an integrity algorithm.

Click here to view code image

```
Router(config)#crypto ikev2 proposal ikev2_proposal
Router(config-ikev2-proposal)#encryption aes-gcm-128 ?
  aes-gcm-256  Combined-mode,256 bit key,16 byte ICV(Authentication Tag)
  <cr>

Router(config-ikev2-proposal)#encryption aes-gcm-256 aes-gcm-128 ?
  <cr>

Router(config-ikev2-proposal)#integrity sha512
 Warning!! Integrity algorithms are not allowed with AES_GCM
Router(config-ikev2-proposal)#group 21 20
Router(config-ikev2-proposal)#prf sha512 sha384
Router(config-ikev2-proposal)#
```

The configured proposal can be seen in running configuration and in the show command output:

Click here to view code image

```
Router#show run | begin ikev2 proposal
crypto ikev2 proposal ikev2_proposal
```

```
 encryption aes-gcm-256 aes-gcm-128
 prf sha512 sha384
 group 21 20

Router#show crypto ikev2 proposal ikev2_proposal
IKEv2 proposal: ikev2_proposal
     Encryption : AES-GCM-256 AES-GCM-128
     Integrity  : none
     PRF        : SHA512 SHA384
     DH Group   : DH_GROUP_521_ECP/Group 21 DH_GROUP_384_ECP/Group 20
```

Configuring IKEv2 Encryption

The encryption algorithm is used for data confidentiality; the algorithm is used to encrypt the IKEv2 Security Association. Combined-mode ciphers give the benefit of authenticated encryption, providing both confidentiality and integrity services using a single algorithm. Table 5-2 displays the available encryption algorithms on Cisco IOS.

Algorithm	Block Size	Key Size	Strength	Quantum Resistant
AES-GCM-256	128	256	NGE	Yes
AES-GCM-128	128	128	NGE	No
AES-CBC-256	128	256	Acceptable	Yes
AES-CBC-192	128	192	Acceptable	No
AES-CBC-128	128	128	Acceptable	No
3DES	64	112	Legacy	No
DES	64	56	Avoid	No

Table 5-2 *IKEv2 Encryption Algorithms Available in Cisco IOS*

The following terms are used to describe the cryptographic strength throughout the rest of this chapter.

- **Avoid:** Algorithms that are marked as Avoid do not provide adequate security against modern threats and should not be used to protect sensitive information. It is recommended that these algorithms be replaced with stronger algorithms.
- **Legacy:** Legacy algorithms provide a marginal but acceptable security level. They should be used only when no better alternatives are available, such as when interoperating with legacy equipment. It is recommended that these legacy algorithms be phased out and replaced with stronger algorithms.
- **Acceptable:** Acceptable algorithms provide adequate security.
- **Next-generation encryption (NGE):** NGE algorithms are expected to meet the security and scalability requirements of the next two decades.

Quantum computer resistant (QCR): Although practical Quantum Computers (QCs) would pose a threat to crypto standards for asymmetric-based cryptographic key exchange and encryption, no one has demonstrated a practical quantum computer yet. It is an area of active research and rapidly growing interest. Although it is possible, it can't be said with certainty whether practical QCs will be built in the future. An algorithm that would be secure even after a QC is built is said to have postquantum security or be quantum computer resistant (QCR). AES-256, SHA-384, and SHA-512

are believed to have postquantum security. There are public key algorithms that are believed to have postquantum security too, but there are no standards for their use in IKEv2 currently. Diffie-Hellman is not QCR.

AES CBC mode supports three key sizes: 128 bits, 192 bits, and 256 bits. AES uses a different number of rounds for each of the defined key sizes. When a 128-bit key is used, 10 rounds are used. When a 192-bit key is used, 12 rounds are used, and when a 256-bit key is used, 14 rounds are used.

AES GCM mode combines both encryption and authentication; for authentication a 16-byte ICV is used as described in RFC5282. The following output shows configuring encryption in IKEv2 proposal

Click here to view code image

```
Router(config-ikev2-proposal)#encryption ?
  3des         3DES
  aes-cbc-128  AES-CBC-128
  aes-cbc-192  AES-CBC-192
  aes-cbc-256  AES-CBC-256
  aes-gcm-128  Combined-mode,128 bit key,16 byte ICV(Authentication Tag)
  aes-gcm-256  Combined-mode,256 bit key,16 byte ICV(Authentication Tag)
  des          DES
```

Note that AES-GCM-128 and AES-GCM-256 are combined-mode ciphers, and if they are used no integrity algorithm is required. These also cannot be used with DES, 3DES, or any AES-CBC mode ciphers.

> **Note**
>
> At the time of writing, the Cisco recommendation is to use AES where possible for encryption of IKE sessions, with AES-GCM preferred over AES-CBC mode. For the latest recommendations, please refer to the Cisco Next Generation Encryption (NGE) page at http://www.cisco.com/c/en/us/about/security-center/next-generation-cryptography.html.

Configuring IKEv2 Integrity

The integrity algorithms will be used to provide assurance that data was not changed in transit. This is achieved by calculating an Integrity Checksum Value (ICV) covering the IKE header and the encrypted payload. The ICV is included within the IKEv2 encrypted payload; this value ensures that any part of the IKE header and the encrypted payload cannot be changed while the packet is in transit. Table 5-3 displays the available integrity algorithms on Cisco IOS.

Algorithm	Checksum Size (bits)	RFC	Strength	Quantum Resistant
Sha521	256	RFC4868	NGE	Yes
Sha384	192	RFC4868	NGE	Yes
Sha256	128	RFC4868	NGE	No
Sha1	96	RFC2404	Legacy	No
Md5	96	RFC2403	Avoid	No

Table 5-3 *IKEv2 Integrity Algorithms Available in Cisco IOS*

> **Note**
>
> Cisco recommends using a SHA2 family Integrity algorithm where possible.

The larger integrity size ensures greater security, but at the cost of a larger overhead for processing and the Integrity Checksum. The following command shows the configuration of integrity algorithms in the IKEv2 proposal.

Click here to view code image

```
Router(config-ikev2-proposal)#integrity sha384 sha256 sha1
```

The integrity algorithms will all be sent by the initiator in that order within the IKE_SA_INIT exchange contained with the transform that relates to the proposal. For a device that is acting as a responder, the integrity algorithms will be chosen in the order configured.

Configuring IKEv2 Diffie-Hellman

The Diffie-Hellman group denotes the size and type of the Diffie-Hellman exchange, in which a shared secret is exchanged by both parties over an insecure medium.

With regards to the Diffie-Hellman group, if the device is acting as the initiator of the IKEv2 session, then the Diffie-Hellman public value sent will always be the first group configured within the proposal. If the initiator of an IKEv2 session is using the default IKEv2 proposal, then the Diffie-Hellman group 5 public value will always be sent in the initial exchange. For the sake of efficiency, the correct Diffie-Hellman group should be configured whenever possible, because an incorrect Diffie-Hellman value will incur an additional two packet exchange to the IKE_SA_INIT.

> **Note**
>
> Cisco recommends using Diffie-Hellman group 14 or higher, with the Elliptic Curve (EC) Diffie-Hellman groups (group 19, 20, and 21) being preferred. Some platforms perform Diffie-Hellman calculations in software; this should be taken into account when devices will be terminating a number of IKEv2 sessions and performing a large number of calculations, because servicing these calculations can overload the CPU and disrupt other processes that require it.

The following output shows configuring Diffie-Hellman groups in an IKEv2 proposal. Four Diffie-Hellman groups are configured, with group 20 being used for any IKEv2 sessions initiated from this device; however if the configured proposal is used when the device is acting as a responder, then the configured groups will be used to match the group proposed by the initiator.

Click here to view code image

```
Router(config-ikev2-proposal)#group 20 21 19 16
```

Table 5-4 displays the available Diffie-Hellman groups on Cisco IOS.

Diffie Hellman Group	RFC	Strength
1 DH 768 MODP	RFC7296	Avoid
2 DH 1024 MODP	RFC7296	Avoid
5 DH 1536 MODP	RFC7296	Avoid
14 DH 2048 MODP	RFC3526	Acceptable
15 DH 3072 MODP	RFC3526	Acceptable
16 DH 4096 MODP	RFC3526	Acceptable
19 DH 256 ECP	RFC5903	Acceptable
20 DH 384 ECP	RFC5903	NGE
21 DH 521 ECP	RFC5903	NGE
24 DH 2048 (256 subgroup) MODP	RFC5114	Acceptable

Table 5-4 *IKEv2 Diffie-Hellman Groups Available in Cisco IOS*

Currently no Diffie-Hellman groups are postquantum secure.

Elliptic Curve Diffie-Hellman (ECDH) groups (19, 20, and 21) offer the most security for the amount of work required. The size of the Elliptic Curve Diffie-Hellman groups can't be compared like for like with MODP groups, because the ECDH groups are more efficient.

Configuring IKEv2 Pseudorandom Function

The pseudorandom function (PRF) is used to generate key material from the derived shared secret. The PRF is a keyed-hash message authentication code (HMAC), which is a cryptographic hash function combined with a secret cryptographic key.

When an integrity algorithm is configured, the same algorithm will automatically be used as the PRF algorithm. Interestingly, when configured for integrity algorithms MD5 and SHA1 both give a 96-bit output; however, when used for PRF, these will give 128-bit and 160-bit outputs respectively. The reason being that the output for the integrity algorithms is truncated, however this is not the case for PRF.

The following example shows how to configured three PRF algorithms, which will be selected in the order that they are configured (SHA256, being the higher preference, is listed first).

Click here to view code image

```
Router(config-ikev2-proposal)#prf sha256 sha1 md5
```

Table 5-5 displays the available pseudorandom function algorithms on Cisco IOS.

Algorithm	Output Size (bits)	RFC	Strength
SHA521	256	RFC4868	NGE
SHA384	192	RFC4868	NGE
SHA256	128	RFC4868	NGE
SHA1	160	RFC2104	Legacy
MD5	128	RFC2104	Avoid

Table 5-5 *IKEv2 Pseudorandom Function Algorithms Available in Cisco IOS*

If a combined-mode cipher for encryption is used, then there is no requirement for any integrity algorithms, but in this case a PRF algorithm must be configured.

Default IKEv2 Proposal

The default IKEv2 proposal consists of the most commonly used transform values, as shown below. The default proposal is associated with default IKEv2 policy, which is used in the IKEv2 negotiation only when there is no user-defined policy. The default policy can be disabled or modified to address the specific needs of users, after which it is displayed in the running configuration. The following examples show disabling, modifying, and restoring the default IKEv2 proposal.

Click here to view code image

```
Router#show crypto ikev2 proposal default
 IKEv2 proposal: default
      Encryption : AES-CBC-256 AES-CBC-192 AES-CBC-128
      Integrity  : SHA512 SHA384 SHA256 SHA96 MD596
      PRF        : SHA512 SHA384 SHA256 SHA1 MD5
      DH Group   : DH_GROUP_1536_MODP/Group 5 DH_GROUP_1024_MODP/Group 2

Router#show crypto ikev2 policy default
 IKEv2 policy : default
      Match fvrf : any
      Match address local : any
      Proposal    : default
```

To disable the default IKEv2 proposal, the default IKEv2 policy that references default proposal must be disabled first.

Click here to view code image

```
Router(config)#no crypto ikev2 proposal default
% Cannot remove as proposal is in use.

Router(config)#no crypto ikev2 policy default
Router(config)#no crypto ikev2 proposal default

Router#show run | include ikev2 proposal default
no crypto ikev2 proposal default

Router#show crypto ikev2 proposal default
 IKEv2 proposal: default Disabled
```

The default proposal is re-enabled with the following command:

Click here to view code image

```
Router(config)#default crypto ikev2 proposal

Router#show run | include ikev2 proposal default
Router#show crypto ikev2 proposal default
 IKEv2 proposal: default
     Encryption : AES-CBC-256 AES-CBC-192 AES-CBC-128
     Integrity  : SHA512 SHA384 SHA256 SHA96 MD596
     PRF        : SHA512 SHA384 SHA256 SHA1 MD5
     DH Group   : DH_GROUP_1536_MODP/Group 5 DH_GROUP_1024_MODP/Group 2
```

When modifying the default IKEv2 proposal, note that modified default proposal is displayed in running configuration.

Click here to view code image

```
Router(config)#crypto ikev2 proposal default
%Warning: This will Modify Default IKEv2 Proposal. Exit if you don't want
```

The integrity algorithms must be removed before configuring a combined mode cipher.

Click here to view code image

```
Router(config-ikev2-proposal)#no integrity sha512 sha384 sha256 sha1 md5
Router(config-ikev2-proposal)#encryption aes-gcm-256
Router(config-ikev2-proposal)#prf sha512
Router(config-ikev2-proposal)#group 21
Router(config-ikev2-proposal)#

Router#show run | begin ikev2 proposal default
crypto ikev2 proposal default
 encryption aes-gcm-256
 prf sha512
 group 21

Router#show crypto ikev2 proposal default
 IKEv2 proposal: default
     Encryption : AES-GCM-256
     Integrity  : none
     PRF        : SHA512
     DH Group   : DH_GROUP_521_ECP/Group 21
```

The modified default proposal can be restored as follows:

Click here to view code image

```
Router(config)#default crypto ikev2 proposal
Router#show run | begin ikev2 proposal default
Router#show crypto ikev2 proposal default
 IKEv2 proposal: default
     Encryption : AES-CBC-256 AES-CBC-192 AES-CBC-128
     Integrity  : SHA512 SHA384 SHA256 SHA96 MD596
     PRF        : SHA512 SHA384 SHA256 SHA1 MD5
     DH Group   : DH_GROUP_1536_MODP/Group 5 DH_GROUP_1024_MODP/Group 2
```

IKEv2 Policy

The IKEv2 policy defines the set of IKEv2 proposals used in an IKEv2 negotiation based on the local context determined by the local address and FVRF used by the IKEv2 negotiation. Note that this applies to both outgoing and incoming negotiations.

An IKEv2 policy can be defined to cater to a specific or multiple contexts. In the case of overlapping policies (that is, when multiple IKEv2 policies are available for a given context), the best or most specific match is used, although configuration of overlapping policies is not recommended for simplicity.

The default IKEv2 policy contains the default IKEv2 proposal and matches any context, that is, any FRVF and any IPv4 or IPv6 local address. Note that the default IKEv2 policy is used for IKEv2 negotiation only in the absence of any user-defined IKEv2 policy.

From the IKEv2 RFC perspective, the IKEv2 policy corresponds to the IKEv2 Security Association (SA) payload in the IKE_SA_INIT messages used for IKE SA negotiation that contains one or more proposals.

Note

The following is a quick recap of VRFs, IVRF, and FVRF, which were described in chapter 4, "IOS IPsec Implementation."

- VRF (Virtual Routing and Forwarding) is a Cisco IOS layer-3 device virtualization technique that splits an IOS device (router/L3-switch) into multiple logical devices, each with its own interfaces and forwarding context, and is used for segregation. Cisco IOS uses two types of VRFs: FVRF and IVRF.
- FVRF (front-door VRF) applies to the underlay network and hence to WAN interfaces and traffic that use underlay addresses, such as IKE and encrypted/encapsulated traffic. FVRF is used for segregation of underlay networks, for example, when a router has multiple WAN connections from different providers with overlapping address spaces.
- IVRF (inside VRF) applies to the overlay network and hence to LAN interfaces and traffic that use overlay addresses such as plaintext traffic before encryption/encapsulation and after decryption/de-encapsulation. IVRF is used for segregation of overlays for multitenancy, for example, when LAN side interfaces of a router connect to different customers or departments that can have overlapping address spaces.
- From a VRF perspective, Cisco IOS tunnel interfaces are an exception in that they use both FVRF and IVRF, as tunnel interfaces bridge both overlay and the underlying transport networks.

The following rules apply to the use of an IKEv2 policy:

- An IKEv2 policy must have at least one IKEv2 proposal and can optionally have more proposals and match statements.
- An IKEv2 policy can optionally have one or match statements to specify the context, that is, FVRFs and local addresses for which the policy will be used.

- A policy can have multiple match statements of same and different types. Multiple match statements of the same type are logically ORed, and match statements of different types are logically ANDed. There is no precedence between match statements of different types.
- It is not recommended to configure overlapping policies. However, in case of multiple possible policy matches, the best match will be used.
- The default IKEv2 policy is used only in the absence of user-defined IKEv2 policies.

Configuring an IKEv2 Policy

An IKEv2 policy is created using the command shown below in the router global configuration mode:

Click here to view code image

```
crypto ikev2 policy policy-name
```

Once the policy is created, the router command prompt will change from **(config)** to **(config-ikev2-policy),** denoting that the user is in IKEv2-policy sub-mode, and a message is displayed stating the requirements for a policy to be complete. The following examples show the creation of an IKEv2 policy.

Click here to view code image

```
Router(config)#crypto ikev2 policy ikev2_policy
IKEv2 policy MUST have at least one complete proposal attached
Router(config-ikev2-policy)#
```

Under the IKEv2-policy sub-mode, IKEv2 proposals and match statements can be configured.

Click here to view code image

```
Router(config-ikev2-policy)#?
IKEv2 Policy commands:
  exit      Exit from IKEv2 policy configuration mode
  match     Match values of local fields
  no        Negate a command or set its defaults
  proposal  Specify Proposal
```

An IKEv2 policy must contain at least one IKEv2 proposal, without which the policy is marked incomplete in the running configuration and is not used in the negotiation.

Click here to view code image

```
Router#show run | begin ikev2 policy
crypto ikev2 policy temp-ikev2-policy1
 ! Policy Incomplete(MUST have at least one complete proposal attached)
```

Configuring IKEv2 Proposals under IKEv2 Policy

Under IKEv2 policy, one or more IKEv2 proposals can be configured that have been defined earlier and are complete. The following output shows configuring proposals under the IKEv2 policy. Note that nonexistent and incomplete proposals are not allowed, and if they are entered, the following error messages are displayed.

Click here to view code image

```
Router(config)#crypto ikev2 policy ikev2_policy
Router(config-ikev2-policy)#proposal ikev2_proposal1
% No Proposal exists with the specified name ikev2_proposal1
```

```
Router(config)#crypto ikev2 proposal ikev2_proposal1
IKEv2 proposal MUST either have a set of an encryption algorithm other than
aes-gcm, an integrity algorithm and a DH group configured or
 encryption algorithm aes-gcm, a prf algorithm and a DH group configured

Router(config)#crypto ikev2 policy ikev2_policy
Router(config-ikev2-policy)#proposal ikev2_proposal1
Proposal ikev2_proposal1 is not complete to be attached to policy
```

Once complete, IKEv2 proposals can be configured under IKEv2 policy

Click here to view code image

```
Router(config)#crypto ikev2 policy ikev2_policy
Router(config-ikev2-policy)#proposal ikev2_proposal1
Router(config-ikev2-policy)#proposal ikev2_proposal2
```

Configuring Match Statements under IKEv2 Policy

One or more match statements can be optionally configured under IKEv2 policy to define the scope of the policy, that is, to specify the FVRFs and the local addresses for which the policy can be used.

The following rules apply to the use of match statements:

- When there are multiple match statements of the same type (either specifying FRVF or local address), the policy is selected when any of the match statements is true. In other words, the match statements of same type have a logical OR relationship.
- When there are multiple match statements of different types (specifying FRVF and local address), the policy is selected when all of the match statements are true. In other words, match statements of different types have a logical AND relationship.
- When there are multiple match statements of both the same and different types, the policy is selected when at least one match statement of each type is true.
- When there are no match statements, the policy can be used with all local addresses in Global VRF.

The following output shows the configuration of match statements under IKEv2 policy.

Click here to view code image

```
Router(config)#crypto ikev2 policy ikev2_policy
Router(config-ikev2-policy)#match ?
  address  Specify the address to match
  fvrf     Specify fvrf

Router(config-ikev2-policy)#match fvrf ?
  WORD  fvrf name (default:global)
  any   Any fvrf

Router(config-ikev2-policy)#match address ?
  local  Local address

Router(config-ikev2-policy)#match address local ?
  A.B.C.D     IPv4 address
  X:X:X:X::X  IPv6 address
```

The following example illustrates configuring an IKEv2 policy with multiple match statements. This

policy would match IKEv2 negotiations that use the FVRF with the name *mpls* or *internet* and use either *192.168.1.1* or *2001::1* as the local address.

Click here to view code image

```
Router(config-ikev2-policy)#match fvrf mpls
Router(config-ikev2-policy)#match fvrf internet
Router(config-ikev2-policy)#match address local 192.168.1.1
Router(config-ikev2-policy)#match address local 2001::1
```

The configured match statements can be seen in running configuration and in the **show** command output.

Click here to view code image

```
Router#show run | begin ikev2 policy
crypto ikev2 policy ikev2_policy
 match fvrf mpls
 match fvrf internet
 match address local 192.168.1.1
 match address local 2001::1
 proposal ikev2_proposal1
 proposal ikev2_proposal2

Router#show crypto ikev2 policy ikev2_policy
 IKEv2 policy : ikev2_policy
      Match fvrf : mpls
      Match fvrf : internet
      Match address local : 192.168.1.1
      Match address local : 2001::1
      Proposal    : ikev2_proposal1
      Proposal    : ikev2_proposal2
```

Default IKEv2 Policy

The default IKEv2 policy contains the default IKEv2 proposal, matches any local address and any FVRF, and is used in the negotiation only when there is no user-defined policy. The default policy can be disabled or modified to address the specific needs of users, after which it is displayed in the running configuration. The following example shows disabling, modifying, and restoring the default IKEv2 policy.

Click here to view code image

```
Router#show crypto ikev2 policy default
 IKEv2 policy : default
      Match fvrf : any
      Match address local : any
      Proposal    : default
```

The default IKEv2 policy is disabled using the following command:

Click here to view code image

```
Router(config)#no crypto ikev2 policy default

Router#show run | include ikev2 policy default
no crypto ikev2 policy default

Router#show crypto ikev2 policy default
 IKEv2 policy : default Disabled
```

The default policy is re-enabled using the following command:

Click here to view code image

```
Router(config)#default crypto ikev2 policy
```

The default IKEv2 policy is modified as shown below, after which it is displayed in running configuration.

Click here to view code image

```
Router(config)#crypto ikev2 policy default
Router(config-ikev2-policy)#match fvrf mpls
% Specific vrf cannot be configured when fvrf type any is already configured.
Remove any fvrf and then add specific
```

The match fvrf any statement in the default IKEv2 policy that matches all the FVRFs must first be removed before a match statement for a specific FRVF is configured.

Click here to view code image

```
Router(config-ikev2-policy)#no match fvrf any
Router(config-ikev2-policy)#match fvrf mpls

Router#show run | begin ikev2 policy default
crypto ikev2 policy default
 match fvrf mpls
 proposal default

Router#show crypto ikev2 policy default
 IKEv2 policy : default
      Match fvrf : mpls
      Match address local : any
      Proposal    : default
```

The modified default policy can be restored as follows:

Click here to view code image

```
Router(config)#default crypto ikev2 policy
```

IKEv2 Policy Selection on the Initiator

The IKEv2 policy selection on the initiator of an outgoing IKEv2 session happens before the negotiation starts, that is, before the IKE_SA_INIT request is sent, as described below.

IKEv2 initiates a session with a peer when IPsec requests IKEv2 to establish an IPsec Security Association with that peer. IPsec Security Associations originate from IPsec interfaces, that is, the interfaces on which IPsec is configured. With IKEv2 the IPsec interface is typically a tunnel interface with the **tunnel protection IPsec** command applied, so it is the configuration on the tunnel interface that determines the outgoing IKEv2/IPsec session parameters as described below;

- The session local address is derived from the **tunnel source** configuration. This typically is the address of the WAN interface through which IKEv2 packets enter and exit the router, but it can also be the address of a loopback interface.
- The session remote address is derived either from the **tunnel destination** configuration or is learned dynamically.
- The session IVRF and FVRF are derived from the **tunnel fvrf** configuration, and this must be

same as the VRF configured on the WAN interface through which IKEv2 packets enter and exit the router. Note that the default value for **tunnel fvrf** is the global routing table (GRT), and in this case IVRF = FVRF = global routing table.

The session parameters derived from the tunnel interface are passed by IPsec to IKEv2 as part of the SA request, and IKEv2 uses these parameters to select an IKEv2 policy to use for the outgoing negotiation based on the match criteria specified in the policy.

The following output shows the IKEv2 policy selection on an initiator, based on the session parameters passed by IPsec. The IKEv2 policy selection happens before the IKE_SA_INIT request is sent to the peer. The highlighted output shows the policy lookup based on FVRF and local address. The output is obtained from the tunnel interface configuration and from **debug crypto ikev2** and **debug crypto kmi** logs.

Click here to view code image

```
interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 tunnel source 192.168.1.1
 tunnel mode ipsec ipv4
 tunnel destination 192.168.1.2
 tunnel protection ipsec profile default

KMI: (Session ID: 3) IPSEC key engine sending message KEY_ENG_REQUEST_SAS to
Crypto IKEv2.
KMI: (Session ID: 3) Crypto IKEv2 received message KEY_ENG_REQUEST_SAS from IPSEC
key engine.
IKEv2:Searching Policy with fvrf 0, local address 192.168.1.1
IKEv2:Using the Default Policy for Proposal
IKEv2:Found Policy 'default'
IKEv2:(SESSION ID = 3,SA ID = 1):Generating IKE_SA_INIT message
IKEv2:(SESSION ID = 3,SA ID = 1):Sending Packet [To 192.168.1.2:500/From
192.168.1.1:500/VRF i0:f0]
Initiator SPI : CF2306A9194583C6 - Responder SPI : 0000000000000000 Message id: 0
IKEv2 IKE_SA_INIT Exchange REQUEST
```

If a matching IKEv2 policy is not found, the following **debug crypto ikev2** log message is displayed.

Click here to view code image

```
IKEv2:Searching Policy with fvrf 0, local address 192.168.1.1
IKEv2-ERROR:Failed to initiate sa
```

IKEv2 Policy Selection on Responder

The IKEV2 policy selection on the responder for an incoming IKEv2 session happens after the IKE_SA_INIT request is received. The FVRF used for policy selection is derived from the VRF of the interface through which the IKEv2 packet entered the router, and the local address is derived from the destination address in the received IKEv2 packet. The IKEv2 policy matching is based on the FVRF, the local address thus derived, and the proposals in the IKE_SA_INIT request.

The following output shows IKEv2 policy selection on a responder when the first IKEv2 packet IKE_SA_INIT request is received, using selected output from **debug crypto ikev2**, logs to show the IKEv2 policy selection. The highlighted output shows policy lookup.

Click here to view code image

```
IKEv2:Received Packet [From 192.168.1.1:500/To 192.168.1.2:500/VRF i0:f0]
```

```
Initiator SPI : C51E7F53FCA49E64 - Responder SPI : 0000000000000000 Message id: 0
IKEv2 IKE_SA_INIT Exchange REQUEST
Payload contents:
 SA KE N VID VID NOTIFY(NAT_DETECTION_SOURCE_IP)
NOTIFY(NAT_DETECTION_DESTINATION_IP)
IKEv2:(SESSION ID = 850,SA ID = 1):Verify SA init message
IKEv2:(SESSION ID = 850,SA ID = 1):Insert SA
IKEv2:Searching Policy with fvrf 0, local address 192.168.1.2
IKEv2:Using the Default Policy for Proposal
IKEv2:Found Policy 'default'
```

If a matching IKEv2 policy is not found, the following **debug crypto ikev2** log messages are displayed. The reason for the 'policy not found' error is mismatching or incorrect IKEv2 policy configuration on the peers.

Click here to view code image

```
IKEv2:Received Packet [From 192.168.1.1:500/To 192.168.1.2:500/VRF i0:f0]
Initiator SPI : 418BD852B092D462 - Responder SPI : 0000000000000000 Message id: 0
IKEv2 IKE_SA_INIT Exchange REQUEST
Payload contents:
 SA KE N VID VID NOTIFY(NAT_DETECTION_SOURCE_IP)
NOTIFY(NAT_DETECTION_DESTINATION_IP)

IKEv2:(SESSION ID = 851,SA ID = 1):Verify SA init message
IKEv2:(SESSION ID = 851,SA ID = 1):Insert SA
IKEv2:Searching Policy with fvrf 0, local address 192.168.1.2
IKEv2-ERROR:(SESSION ID = 851,SA ID = 1):: Failed to locate an item in the
database
IKEv2:(SESSION ID = 851,SA ID = 1):Failed SA init exchange
IKEv2-ERROR:(SESSION ID = 851,SA ID = 1):Initial exchange failed: Initial exchange
failed
IKEv2:(SESSION ID = 851,SA ID = 1):Abort exchange
IKEv2:(SESSION ID = 851,SA ID = 1):Deleting SA
```

IKEv2 Policy Configuration Examples

Per-peer IKEv2 Policy

As IKEv2 policy selection is based on local address and FVRF and not remote IP address, different polices for different peers can be achieved by using different local address or FVRF for connecting to those peers.

This was an actual requirement where a customer needed to allow connections to a router from two different IKEv2 peers (called Peer1 and Peer2) and use specific Next Generation Encryption (NGE) algorithms only for connections from Peer1 and not Peer2. This required two separate local IP addresses that were configured as the tunnel destination address on the two peers. The unique local IP addresses were used to differentiate two separate Policies (Policy1 and Policy2), connections from Peer1 would be anchored on the IP address defined within Policy1 (192.168.1.1) and connections from Peer2 would be anchored on the IP address defined within Policy2 (192.168.2.1). The relevant IKEv2 proposals were defined within each seperate IKEv2 policy, which then meets the requirement that only Proposal1 will be used for IKEv2 connections anchored on 192.168.1.1 and Proposal2 will be used for IKEv2 connections anchored on 192.168.2.1. The following example illustrates an IKEv2 policy configuration, which is used for two peers.

Click here to view code image

```
Router#show crypto ikev2 proposal
 IKEv2 proposal: Proposal1
     Encryption : AES-GCM-128
     Integrity  : none
     PRF        : SHA256
     DH Group   : DH_GROUP_256_ECP/Group 19
 IKEv2 proposal: Proposal2
     Encryption : AES-CBC-128
     Integrity  : SHA96
     PRF        : SHA1
     DH Group   : DH_GROUP_2048_MODP/Group 14

Router#show crypto ikev2 policy
 IKEv2 policy : Policy1
      Match fvrf  : global
      Match address local : 192.168.1.1
      Proposal    : Proposal1
 IKEv2 policy : Policy2
      Match fvrf  : global
      Match address local : 192.168.2.1
      Proposal    : Proposal2
```

IKEv2 Policy with Multiple Proposals

The following example shows how two IKEv2 proposals can be used within an IKEv2 policy. This would be required if a device is serving both combined and non-combined mode ciphers when establishing IKEv2 Security Associations.

Click here to view code image

```
Router#show crypto ikev2 proposal
 IKEv2 proposal: Proposal1
     Encryption : AES-GCM-128
     Integrity  : none
     PRF        : SHA256
     DH Group   : DH_GROUP_256_ECP/Group 19
 IKEv2 proposal: Proposal2
     Encryption : AES-CBC-128
     Integrity  : SHA96
     PRF        : SHA1
     DH Group   : DH_GROUP_2048_MODP/Group 14

Router#show crypto ikev2 policy
 IKEv2 policy : default
      Match fvrf : any
      Match address local : any
      Proposal    : Proposal1
      Proposal    : Proposal2
```

The following debug log, enabled using the command **debug crypto ikev2 packet**, shows both Proposals being sent when the device is initiating an IKEv2 Security Association. The Diffie-Hellman public value is selected from the preferred Proposal *Proposal1*.

Click here to view code image

```
IKEv2:(SESSION ID = 1,SA ID = 1):Generating IKE_SA_INIT message
IKEv2:(SESSION ID = 1,SA ID = 1):IKE Proposal: 1, SPI size: 0 (initial
negotiation),
Num. transforms: 3
   AES-GCM   SHA256   DH_GROUP_256_ECP/Group 19
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):IKE Proposal: 2, SPI size: 0 (initial
negotiation),
Num. transforms: 4
   AES-CBC   SHA1   SHA96   DH_GROUP_2048_MODP/Group 14

IKEv2:(SESSION ID = 1,SA ID = 1):Next payload: SA, version: 2.0 Exchange type:
IKE_SA_INIT, flags: INITIATOR Message id: 0, length: 308
Payload contents:
 SA  Next payload: KE, reserved: 0x0, length: 84
  last proposal: 0x2, reserved: 0x0, length: 36
  Proposal: 1, Protocol id: IKE, SPI size: 0, #trans: 3    last transform: 0x3,
reserved: 0x0: length: 12
    type: 1, reserved: 0x0, id: AES-GCM
    last transform: 0x3, reserved: 0x0: length: 8
    type: 2, reserved: 0x0, id: SHA256
    last transform: 0x0, reserved: 0x0: length: 8
    type: 4, reserved: 0x0, id: DH_GROUP_256_ECP/Group 19
  last proposal: 0x0, reserved: 0x0, length: 44
  Proposal: 2, Protocol id: IKE, SPI size: 0, #trans: 4    last transform: 0x3,
reserved: 0x0: length: 12
    type: 1, reserved: 0x0, id: AES-CBC
    last transform: 0x3, reserved: 0x0: length: 8
    type: 2, reserved: 0x0, id: SHA1
    last transform: 0x3, reserved: 0x0: length: 8
    type: 3, reserved: 0x0, id: SHA96
    last transform: 0x0, reserved: 0x0: length: 8
    type: 4, reserved: 0x0, id: DH_GROUP_2048_MODP/Group 14
 KE  Next payload: N, reserved: 0x0, length: 72
    DH group: 19, Reserved: 0x0
```

We then see that the responder has accepted the algorithms contained within Proposal1 and replies with these as the selected algorithms within the Proposal. We also can see that the Diffie-Hellman public value is included, which matches the same group used by the initiator.

Click here to view code image

```
IKEv2:(SESSION ID = 1,SA ID = 1):Next payload: SA, version: 2.0 Exchange type:
IKE_SA_INIT, flags: RESPONDER MSG-RESPONSE Message id: 0, length: 264
Payload contents:
 SA  Next payload: KE, reserved: 0x0, length: 40
  last proposal: 0x0, reserved: 0x0, length: 36
  Proposal: 1, Protocol id: IKE, SPI size: 0, #trans: 3    last transform: 0x3,
reserved: 0x0: length: 12
    type: 1, reserved: 0x0, id: AES-GCM
    last transform: 0x3, reserved: 0x0: length: 8
    type: 2, reserved: 0x0, id: SHA256
    last transform: 0x0, reserved: 0x0: length: 8
    type: 4, reserved: 0x0, id: DH_GROUP_256_ECP/Group 19
 KE  Next payload: N, reserved: 0x0, length: 72
    DH group: 19, Reserved: 0x0
```

IKEv2 Keyring

Authentication within IKEv2 is accomplished by three methods: public key signatures, EAP, and shared secret. The terms shared secret and pre-shared key can be used interchangeably.

Within IKEv2, authentication is performed asymmetrically: that is, one device can use one method of authentication to prove its identity, and the peer device can use another method. When shared secrets are used, the identity of a device is proven by an initiator by sending the following output of the PRF

within the AUTH payload in IKE_AUTH messages. The same concept is used when the responder proves its identity.

Click here to view code image

```
AUTH = prf( prf(Shared Secret, "Key Pad for IKEv2"), <InitiatorSignedOctets>)
```

If the initiator's configured shared secret for the peer it was connecting to was *'Cisco123'*, then the AUTH payload would be constructed as;

Click here to view code image

```
AUTH = prf( prf(Cisco123, "Key Pad for IKEv2"), <InitiatorSignedOctets>)
```

The following displays an AUTH payload that uses shared secrets. The authentication data is the output of the PRF and can be seen to be 160 bits, an indication that SHA1 was used as the PRF.

Click here to view code image

```
Type Payload: Authentication (39)
                    Next payload: Security Association (33)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 28
                    Authentication Method: Shared Key Message Integrity Code (2)
                    Authentication Data: 27c5615a79ba2abe1c0077507770d624c3f8e246
```

On receipt of this payload, the peer can reconstruct the output of the PRF itself, for the PRF was previously negotiated in the IKE_SA_INIT exchange, the string "Key Pad for IKEv2" is known, and the 'InitiatorSignedOctets' is also available to the receiver. The pre-shared key for the peer is looked up based on the peer IP address or an IKE identity from the keyring. Once the pre-shared key has been located, all components are then available to reconstruct the authentication data. If the locally generated authentication matches the authentication data received, then the responder can be sure that the peer device is in possession of the pre-shared key and hence is authenticated.

The IKEv2 keyring is a local repository of pre-shared keys that are used with the shared secret authentication method. Note that the term local repository is used to highlight the fact that keys can also be stored remotely on a RADIUS server. One or more keyrings can be defined, and the keyrings must be referenced from the IKEv2 profile in order to be used for key lookup.

The keyring is structured into peer blocks in which each block allows grouping of one or more related peers based on their IP addresses, IKE identities, and hostnames and then defines the pre-shared key to be used with those peers. The key defined in a peer block can be either symmetric or asymmetric. A symmetric key is bidirectional in that the same key is used by a device to authenticate itself to a peer as well as to authenticate that peer (that is, to generate the authentication data sent to a peer as well as to verify the authentication data sent by that peer, which must be configured with the same key). An asymmetric key is unidirectional in that it is used by a device to either authenticate itself to a peer, in which case the key is referred to as local key, or to authenticate a peer in which case the key is referred to as remote key. The local key on a device must be configured as a remote key on the peer and vice versa.

Pre-shared keys can also be stored on a RADIUS server, which offers a scalable and centralized way of managing pre-shared keys. This is especially true for IKEv2 nodes that need to support pre-shared key authentication with a large number of peers, such as a FlexVPN Server acting as a hub or remote access server. The keys are obtained from the server using RADIUS authorization, which is based on

peer IKE identity, and since identity-based key lookup is possible only on an IKEv2 responder, RADIUS-based pre-shared keys can be used only on IKEv2 responders.

Since RADIUS-based pre-shared keys are particularly relevant to the FlexVPN Server, this topic will be described in detail in chapter 9, "FlexVPN Server."

Configuring IKEv2 Keyring

An IKEv2 keyring is created using the following command in the router's global configuration mode:

Click here to view code image

```
crypto ikev2 keyring keyring-name
```

The following examples show the creation of an IKEv2 keyring. Once the keyring is created, the router command prompt will change to **(config-ikev2-keyring)**, denoting that user is in IKEv2-keyring sub-mode.

Click here to view code image

```
Router(config)#crypto ikev2 keyring ikev2_keyring
Router(config-ikev2-keyring)#
```

The keyring must be referenced from the IKEv2 profile in order to be used for key lookup and is used for lookup only for peers associated with that IKEv2 profile. Note that only one keyring can be referenced from a profile.

Click here to view code image

```
Router(config)#crypto ikev2 profile default
Router(config-ikev2-profile)#keyring local ikev2_keyring
```

The keyring must be dereferenced from the IKEv2 profile before the keyring is deleted.

Click here to view code image

```
Router(config)#no crypto ikev2 keyring ikev2_keyring
% Cannot remove as keyring is in use.
Router(config)#crypto ikev2 profile default
Router(config-ikev2-profile)#no keyring local ikev2_keyring
Router(config-ikev2-profile)#exit
Router(config)#no crypto ikev2 keyring ikev2_keyring
Router(config)#
```

Configuring a Peer Block in Keyring

The following example shows the creation of a peer block under an IKEv2 keyring.

Click here to view code image

```
Router(config-ikev2-keyring)#?
IKEv2 Keyring commands:
  exit  Exit from crypto ikev2 keyring sub-mode
  no    Negate a command or set its defaults
  peer  Configure a Peer and associated keys
```

Once a peer block is created, the router command prompt will change to **(config-ikev2-keyring-peer)**, denoting that the user is in the peer sub-mode. Under the peer sub-mode, peers and their corresponding pre-shared key can be configured

Click here to view code image

```
Router(config-ikev2-keyring)#peer peer1
Router(config-ikev2-keyring-peer)#?
Crypto IKEv2 Keyring Peer sub-mode commands:
  address          Specify IPv4/IPv6 address of peer
  description      Specify a description of this peer
  exit             Exit from crypto ikev2 keyring peer sub-mode
  hostname         Hostname of peer
  identity         Specify IKE identity to use
  no               Negate values of a command
  pre-shared-key   specify the pre-shared key
```

Configuring Peers in a Peer Block

In the keyring peer block, peers can be specified by their IP address, hostname, and identity; only one statement of each type is allowed. The IP address and identity statements can specify one or more peers using the IP address mask and domain portion of the identity (FQDN and email only), but the hostname statement can only specify a single peer. A wildcard key can be configured by specifying peers using a wildcard IP address and mask (0.0.0.0 0.0.0.0); however, the use of wildcard keys is not recommended.

The key lookup can be based on any of the three parameters—hostname, IP address, or identity—to find a matching peer block and the corresponding key. In case of overlapping peer blocks, the best match is used, though such a configuration is not recommended for simplicity.

The key lookup by peer hostname is performed only by the hostname string and not by its resolved address, as address resolution is not performed during the lookup. The key lookup by peer hostname has very limited use; although the key lookup by peer hostname can be performed on an initiator, specifying the IKE/IPsec peer as a hostname is possible only with crypto map configuration by use of the **set peer hostname dynamic** command and not with a tunnel interface configuration. The following output shows specifying peers in a peer block.

Specifying peers using their IP address and mask:

Click here to view code image

```
Router(config-ikev2-keyring-peer)#address ?
  A.B.C.D              IPv4 Address
  X:X:X:X::X/<0-128>   IPv6 address/prefix

Router(config-ikev2-keyring-peer)#address 192.168.1.0 ?
  A.B.C.D  specify mask
  <cr>

Router(config-ikev2-keyring-peer)#address 2001::1/128
```

Specifying a peer using hostname,

Click here to view code image

```
Router(config-ikev2-keyring-peer)#hostname ?
  WORD  Name
```

Specifying peers using their Identity,

Click here to view code image

```
Router(config-ikev2-keyring-peer)#identity ?
```

```
  address  IP address
  email    Use email address
  fqdn     Use FQDN
  key-id   proprietary types of identification (ID KEY ID)

Router(config-ikev2-keyring-peer)#identity email ?
  WORD    Specify the name string
  domain  email Domain Name

Router(config-ikev2-keyring-peer)#identity email domain ?
  WORD  Specify the name string

Router(config-ikev2-keyring-peer)#identity fqdn ?
  WORD    Specify the name string
  domain  email Domain Name

Router(config-ikev2-keyring-peer)#identity fqdn domain ?
  WORD  Specify the name string

Router(config-ikev2-keyring-peer)#identity key-id ?
  WORD  ID_KEY_ID <string>

Router(config-ikev2-keyring-peer)#identity address ?
  A.B.C.D     IPv4 Address
  X:X:X:X::X  IPv6 address
```

Configuring Keys in a Peer Block

A peer block can have a single symmetric key or an asymmetric key pair. An asymmetric key is specified by using an additional **local** or **remote** keyword. With a symmetric key, the peer must be configured with the same key for this device. With asymmetric keys, the local key configured for a peer on one device must be configured as the remote key for this device on the peer and vice versa. The following example shows configuring pre-shared keys in a peer block.

Click here to view code image

```
Router(config-ikev2-keyring-peer)#pre-shared-key ?
  0       Specifies an UNENCRYPTED password will follow
  6       Specifies an ENCRYPTED password will follow
  LINE    The UNENCRYPTED (cleartext) user password
  hex     Key entered in hex string
  local   specify signing key
  remote  specify verifying key

Router(config-ikev2-keyring-peer)#pre-shared-key local ?
  0     Specifies an UNENCRYPTED password will follow
  6     Specifies an ENCRYPTED password will follow
  LINE  The UNENCRYPTED (cleartext) user password
  hex   Key entered in hex string

Router(config-ikev2-keyring-peer)#pre-shared-key remote ?
  0     Specifies an UNENCRYPTED password will follow
  6     Specifies an ENCRYPTED password will follow
  LINE  The UNENCRYPTED (cleartext) user password
  hex   Key entered in hex string
```

Key Lookup on Initiator

The key lookup on the initiator is performed before the start of IKE negotiation; hence the lookup can be based only on peer IP address or hostname, because the peer IKE identity (received in IKE_AUTH response) is not yet available. The key lookup is first performed using hostname (if available) and then by IP address if the key is not found.

Each peer block in an IKEv2 keyring must have pre-shared keys defined for a unique set of peers, and overlapping peer blocks should be avoided. However, in case of multiple matching peer blocks, the most specific match is used.

The following example shows a pre-shared key lookup being performed on the initiator at the start of IKEv2 negotiation before the first IKEv2 packet IKE_SA_INIT request is sent to the peer, using selected output below from **debug crypto ikev2**, **debug crypto ikev2 internal**, and **debug crypto kmi** logs. The highlighted output shows the key lookup.

Click here to view code image

```
KMI: (Session ID: 3) IPSEC key engine sending message KEY_ENG_REQUEST_SAS to
Crypto IKEv2.
KMI: (Session ID: 3) Crypto IKEv2 received message

IKEv2:% Getting preshared key from profile keyring ikev2_keyring
IKEv2-INTERNAL:% Getting preshared key by address 192.168.1.2
IKEv2:% Matched peer block 'Router2'

IKEv2:(SESSION ID = 3,SA ID = 1):Generating IKE_SA_INIT message
IKEv2:(SESSION ID = 3,SA ID = 1):Sending Packet [To 192.168.1.2:500/From
192.168.1.1:500/VRF i0:f0]
Initiator SPI : 83BA3833DB300CF3 - Responder SPI : 0000000000000000 Message id: 0
IKEv2 IKE_SA_INIT Exchange REQUEST
```

If the key is not found, the following error log message is displayed if **debug crypto ikev2** is enabled.

Click here to view code image

```
KMI: (Session ID: 3) IPSEC key engine sending message KEY_ENG_REQUEST_SAS to
Crypto IKEv2.
KMI: (Session ID: 3) Crypto IKEv2 received message KEY_ENG_REQUEST_SAS from IPSEC
key engine.
IKEv2:% Getting preshared key from profile keyring ikev2_keyring
IKEv2-ERROR:% key not found.
IKEv2-ERROR:Failed to initiate sa
```

Key Lookup on Responder

The key lookup on the responder is performed when an IKE_AUTH request is received; hence the lookup can be based on peer IKE identity (received in IKE_AUTH request) or IP address but not peer hostname, for the responder learns the peer from negotiation by its IP address and identity. The key lookup is first performed using the peer identity and then by IP address if a key is not found. While overlapping peer blocks should be avoided, in case of multiple matching peer blocks, the most specific match is used.

The following example shows the key lookup on a responder using selected output from **debug crypto ikev2** logs. The highlighted output shows the key lookup.

Click here to view code image

```
IKEv2:(SESSION ID = 1069,SA ID = 1):Received Packet [From 192.168.1.1:500/To
192.168.1.2:500/VRF i0:f0]
Initiator SPI : 85A08BDDAE7D63EC - Responder SPI : 2DF24BD4A2F7A66C Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST
IKEv2:(SESSION ID = 1069,SA ID = 1):Searching policy based on peer's identity
'192.168.1.1' of type 'IPv4 address'
IKEv2:found matching IKEv2 profile 'default'
IKEv2-INTERNAL:(1): Choosing IKE profile default
IKEv2:% Getting preshared key from profile keyring ikev2_keyring
IKEv2-INTERNAL:% Getting preshared key by address 192.168.1.1
IKEv2:% Matched peer block 'Router2'
```

If the key is not found, the following error log is displayed if **debug crypto ikev2** and **debug crypto ikev2 internal** are enabled.

Click here to view code image

```
IKEv2:(SESSION ID = 1090,SA ID = 1):Received Packet [From 192.168.1.1:500/To
192.168.1.2:500/VRF i0:f0]
Initiator SPI : 16B9C6C2858AC956 - Responder SPI : 386B52AB1240F61D Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST

IKEv2:% Getting preshared key from profile keyring ikev2_keyring
IKEv2:(SESSION ID = 1090,SA ID = 1):Get peer's preshared key for 192.168.1.1
IKEv2-INTERNAL:(SESSION ID = 1090,SA ID = 1):Failed to compute authentication data
IKEv2-ERROR:(SESSION ID = 1090,SA ID = 1):: Failed to authenticate the IKE SA
IKEv2:(SESSION ID = 1090,SA ID = 1):Verification of peer's authentication data
FAILED
IKEv2:(SESSION ID = 1090,SA ID = 1):Sending authentication failure notify
```

Table 5-6 shows the IKEv2 parameters used for pre-shared key lookup and if they can be used on an initiator and/or responder.

Key Lookup Parameter	Initiator or Responder
Peer address (IPv4,IPv6)	Both
Peer identity of type ■ Address (IPv4, IPv6) ■ FQDN ■ Email ■ Key-id	Responder only
Peer Hostname	Initiator and crypto maps only

Table 5-6 *IKEv2 Pre-Shared Key Lookup Parameters*

IKEv2 Keyring Configuration Example

The following examples show a keyring with two peer blocks to highlight the various aspects of keyring configuration.

The first peer block matches peers from company abc.com using their IP addresses and domain portion of their FQDN identities and is configured with asymmetric keys where the local key (key1)

is used to authenticate to specified peers and remote key (key2) is used to validate those peers. The key lookup is first performed using the peer identity if available and if not found, using the peer IP address.

The second peer block matches a specific peer based on its hostname, host IP address, and unique Identity of each type and is configured with a symmetric key, which is used both for authenticating to the peer and validating that peer.

Click here to view code image

```
crypto ikev2 keyring ikev2_keyring
 peer abc.com
  address 192.168.1.0 255.255.255.0
  identity fqdn domain abc.com
  identity email domain abc.com
  pre-shared-key local key1
  pre-shared-key remote key2

 peer peer2
  hostname peer2
  address 192.168.2.1 255.255.255.255
  identity address 192.168.2.1
  identity key-id peer2
  identity fqdn peer2.abc.com
  identity email peer2@abc.com
  pre-shared-key key
```

In the examples, the pre-shared key is cryptographically weak; these values could be trivially broken by someone performing a man-in-the-middle attack (impersonating the responder) on the service-provider network when an initiator initiates an IKEv2 session. For this reason pre-shared keys should be cryptographically strong, containing sufficient entropy to ensure that these cannot be easily broken by a brute-force or dictionary attack. RFC7296 states that the pre-shared-key should be as cryptographically strong as the key being negotiated. If AES is being used with a 128 bit key, then the pre-shared key used should have 128 bits of entropy. If using base94 (which is compatible with the characters that can be entered as a pre-shared key within Cisco IOS), then the key length should be at least 20 characters. Table 5-7 displays cryptographic strengths and the equivalent number of base-94 characters required to achieve this strength.

Cryptographic Strength	Base 94 Characters to Meet Equivalent Strength
128bit	20
192bit	30
256bit	40

Table 5-7 *Cryptographic Strength of Pre-Shared Keys*

Generating random pre-shared keys with sufficient entropy is hard for any human, for human nature gives us tendencies to not create true randomness. Hence passwords should be generated by a device that can generate true randomness. RFC7296 even states the following regarding the use of passwords as pre-shared-keys.

> As noted above, deriving the shared secret from a password is not secure. This construction is used because it is anticipated that people will do it anyway.

IKEv2 Keyring Key Points

Following is a summary of the key points about IKEv2 keyrings.

- One or more keyrings can be defined, and a keyring is used for key lookup during shared-secret authentication only when it is referenced from an IKEv2 profile and is used only for the peers associated with that profile.
- Only one keyring can be referenced from an IKEv2 profile; however, the same keyring can be referenced from more than one profile.
- The key lookup on an initiator is based on the peer's hostname and IP address, while on a responder it is based on the peer's identity and IP address in that order.
- Each peer block in an IKEv2 keyring must have pre-shared keys defined for a unique set of peers, and overlapping peer blocks should be avoided. However, in case of multiple matching peer blocks, the most specific match is used. Symmetric key is used for bidirectional authentication (that is, both for signing and validation of authentication data) and must be configured on both peers.
- Asymmetric key is used for unidirectional authentication either for signing (local key) or validation (remote key) of authentication data. A local key on a device must be configured as the remote key on the peer and vice versa.

IKEv2 Profile

The IKEv2 profile is the central Cisco IOS IKEv2 configuration that defines a peer group based on peer identities and defines various parameters and features to be used with that peer group.

Table 5-8 lists the key IKEv2 profile parameters and the functionality they provide

IKEv2 Profile Parameter	Functionality Provided
Match statements	Group peers based on identity Define scope of the profile
Authentication methods	Local and remote authentication methods
Keyring	Credentials for pre-shared key authentication method
PKI trustpoints	Credentials for certificate-based authentication methods
Local identity	Identity used by local device
AAA authorization	Authorization type, server and username
Config exchange	Configuration exchange type and behavior
Dead peer detection	DPD method and probe intervals
SA lifetime	Time-based lifetime of the IKEv2 SA
NAT keepalive	NAT keepalive duration
Initial contact	Forced initial contact processing
IVRF	IVRF of the crypto map-based IPsec Security Associations
Virtual-template	Configuration template for IPsec dVTI
Shutdown	Disable and prevent profile from being used
Redirect	Enables IKEv2 redirect feature for load balancing
Reconnect	Enables IKEv2 auto-reconnect feature

Table 5-8 *IKEv2 Profile Parameters and Usage*

The parameters and features defined in the profile such as AAA authorization, IKEv2 redirect and IKEv2 auto-reconnect that are relevant to IKEv2 load balancing and FlexVPN server are described in the respective chapters.

The IKEv2 profile is a mandatory configuration because it defines important parameters, such as authentication method and credentials, without which negotiation cannot start. Multiple IKEv2 profiles can be defined to allow per peer-group parameters and features.

An IKEv2 profile must be attached to an IPsec interface, such as a tunnel interface with tunnel protection applied, on both the initiator and responder. This profile is validated during negotiation to check if the received peer identity matches the profile and if not the negotiation is aborted. As an IPsec interface represents an IPsec SPD (Security Policy database) instance in Cisco IOS, the IKEv2 profile associated with an IPsec interface authorizes the peers that can communicate with that SPD.

IKEv2 Profile as Peer Authorization Database

From RFC4301 (Security Architecture for the Internet Protocol) perspective, IKEv2 profile embodies a Peer Authorization Database (PAD). Here is a description of PAD from the RFC.

Click here to view code image

```
4.4.3.  Peer Authorization Database (PAD)

   The Peer Authorization Database (PAD) provides the link between the
   SPD and a security association management protocol such as IKE.  It
   embodies several critical functions:

        o identifies the peers or groups of peers that are authorized
          to communicate with this IPsec entity
        o specifies the protocol and method used to authenticate each
          peer
        o provides the authentication data for each peer
        o constrains the types and values of IDs that can be asserted
          by a peer with regard to child SA creation, to ensure that the
          peer does not assert identities for lookup in the SPD that it
          is not authorized to represent, when child SAs are created
        o peer gateway location info, e.g., IP address(es) or DNS names,
          MAY be included for peers that are known to be "behind" a
          security gateway

   The PAD provides these functions for an IKE peer when the peer acts
   as either the initiator or the responder.
```

IKEv2 profile implements PAD and provides the functionalities described above:

- Provides a linkage between SPD (IPsec interface) and the key management protocol (IKEv2) as the profile must be attached to an IPsec interface on both initiator and responder.
- Specifies the protocol (IKEv2), authentication method, and credentials.
- Defines the peers that are authorized to communicate with an SPD (IPsec interface).

Configuring IKEv2 Profile

An IKEv2 Profile is created using the following command in the router global configuration mode:

Click here to view code image

```
crypto ikev2 profile profile-name
```

The following examples show IKEv2 profile creation. Once the profile is created, the router command prompt will change to **(config-ikev2-profile)**, denoting that the user is in IKEv2-profile sub-mode, and a message is displayed stating the minimum required configuration in a profile.

Click here to view code image

```
Router(config)#crypto ikev2 profile ikev2_profile
IKEv2 profile MUST have:
   1. A local and a remote authentication method.
   2. A match identity or a match certificate or match any statement.
```

Without the mandatory configuration in a profile, the running configuration displays it as incomplete:

Click here to view code image

```
Router(config)#show run | begin ikev2_profile
crypto ikev2 profile ikev2_profile
 ! Profile incomplete (no match identity or match certificate statement)
 ! Profile incomplete (no local and/or remote authentication method specified)
```

The profile must be attached to a tunnel interface on both the initiator and the responder, as shown in the following example. Only one profile can be attached to an interface. The IKEv2 profile can be

specified with a **tunnel protection** command option or can be configured under an IPsec profile, which in turn is configured on tunnel interface with the **tunnel protection** command.

Click here to view code image

```
Router(config)#interface tunnel 100
Router(config-if)tunnel protection ipsec profile ipsec_profile ikev2-profile
ikev2_profile
```

or

Click here to view code image

```
Router(config)#crypto ipsec profile ipsec_profile
Router(ipsec-profile)#set ikev2-profile ikev2_profile
Router(config)#interface tunnel 100
Router(config-if)tunnel protection ipsec profile ipsec_profile
```

The IKEv2 profile must be detached from IPsec profile before it is deleted:

Click here to view code image

```
Router(config)#no crypto ikev2 profile ikev2_profile
% Profile ikev2_profile is still in use and cannot be removed
Router(config)#crypto ipsec profile ipsec_profile
Router(ipsec-profile)#no set ikev2-profile ikev2_profile
Router(ipsec-profile)#exit
Router(config)#no crypto ikev2 profile ikev2_profile
Router(config)#
```

Configuring Match Statements in IKEv2 Profile

Match statements in an IKEv2 profile serve two purposes: grouping of peers based on their identity and defining the scope of the profile based on the FVRF and local address context of the negotiation. An IKEv2 profile must have at least one statement matching the peer identity (**match identity** or **match certificate**) and can optionally have a statement matching FVRF and one or more statements matching peer identity and local address.

Following rules apply to the use of match statements:

- The **match identity** and **match certificate** statements specify peer identity and hence are considered to be of same type. When both are present, **match certificate** has precedence and is evaluated first if peer certificate is available.
- When there are multiple match statements of the same type (specifying peer identity, FRVF, or local address), the profile is selected when any of the match statements is true. In other words, match statements of the same type have a logical OR relationship.
- When there are multiple match statements of different types (specifying peer identity, FRVF, and local address), the profile is selected when all of the match statements are true. In other words, match statements of different types have a logical AND relationship.
- When there are multiple match statements of both the same and different types, the profile is selected when at least one match statement of each type is true.
- When there are no match statements defining the FVRF, local address, and remote address scope, the profile is used for any local address and global FVRF.

Table 5-9 lists the match statement types supported in IKEv2 profile and the functionality they

provide.

Match Statement	Functionality Provided
Identity	Match peers based on Identity
Certificate	Match peers based on Certificate
FVRF	Define FVRF scope of the profile
Local address	Define local address scope of the profile

Table 5-9 *IKEv2 Profile Match Statement Types and Usage*

Click here to view code image

```
Router(config-ikev2-profile)#match ?
  address      IP address
  certificate  Peer certificate attributes
  fvrf         fvrf of the profile
  identity     IKE identity
```

Matching Peers by Certificate

An IKEv2 profile allows matching peers based on certificate through one or more **match certificate** statements that use PKI Certificate maps to match arbitrary fields in a certificate. When a peer authenticates itself using a certificate, the IKEv2 profile lookup is first performed based on the peer certificate, which is matched against the **match certificate** statements in the configured profiles. If a match is not found, the lookup is then performed by peer identity. The following example shows profile matching by peer certificate.

Click here to view code image

```
Router(config)#crypto pki certificate map certificate_map 1
Router(ca-certificate-map)#?
Crypto certificate attribute map commands:
  alt-subject-name           Alternate subject name
  default                    Set a command to its defaults
  exit                       Exit from certificate attribute map mode
  expires-on                 Expiration date/time
  issuer-name                Certificate issuer name
  name                       Any subject name field
  no                         Negate a command or set its defaults
  serial-number              Certificate serial number (hexadecimal)
  subject-name               Certificate subject name
  unstructured-subject-name  Unstructured subject name
  valid-start                Valid after date/time

Router(ca-certificate-map)#subject-name ?
  co  Contains
  eq  Equal
  nc  Does not contain
  ne  Not Equal

Router(ca-certificate-map)#subject-name eq ?
  LINE  Compare string

Router(ca-certificate-map)#subject-name co Router2
Router(ca-certificate-map)#end
Router#
```

```
Router(config)#crypto ikev2 profile ikev2_profile
Router(config-ikev2-profile)#match certificate certificate_map
Router(config-ikev2-profile)#
```

Matching Peers by Identity

The IKEv2 profile allows matching peers based on identity of all types through one or more **match identity** statements. For structured identity types such as FQDN and email, in addition to exact match, the matching can also be based on the domain portion of the identity, which facilitates easier grouping of peers belonging to a common domain, such as a company or an organization.

For peers authenticating using shared secret and EAP methods, the profile lookup is based on identity, and for peers authenticating using the certificate method, the lookup is first based on certificate and then identity.

Table 5-10 lists the peer Identity types supported for matching the profile.

Peer Identity Type	Description
Address	IPv4 or IPv6 prefix
FQDN	Can be exact match (user.domain) or based on domain
Email	Can be exact match (user@domain) or based on domain
Key-id	Opaque string

Table 5-10 *IKEv2 Peer Identity Types Used for Profile Matching*

The following example shows how to configure profile matching by peer identity:

Click here to view code image

```
Router(config-ikev2-profile)#match identity ?
  remote  Remote identity

Router(config-ikev2-profile)#match identity remote ?
  address  IP Address(es)
  any      match any peer identity
  email    Fully qualified email string [Max. 255 char(s)]
  fqdn     Fully qualified domain name string [Max. 255 char(s)]
  key-id   key-id opaque string

Router(config-ikev2-profile)#match identity remote any ?
  <cr>

Router(config-ikev2-profile)#match identity remote address ?
  A.B.C.D             IP address prefix
  X:X:X:X::X/<0-128>  IPv6 address/prefix-length

Router(config-ikev2-profile)#match identity remote email ?
  WORD    Specify the name string
  domain  match domain instead of exact

Router(config-ikev2-profile)#match identity remote email domain ?
  WORD  Specify the name string

Router(config-ikev2-profile)#match identity remote fqdn ?
  WORD    Specify the name string
  domain  match domain instead of exact
```

```
Router(config-ikev2-profile)#match identity remote fqdn domain ?
  WORD  Specify the name string

Router(config-ikev2-profile)#match identity remote key-id ?
  WORD  Specify the key-id string
```

Matching any Peer Identity

An IKEv2 profile can be configured as a last resort catch-all profile by using a **match identity remote any** statement. This allows the profile to define default parameters and features for peers that do not match any specific profile. Note that the **match identity remote any** statement is mutually exclusive with other **match identity** and **match certificate** statements; however, **match fvrf** and **match address local** statements that define the profile scope do not conflict with the **match identity remote any** statement.

This feature is useful on IKEv2 responders in simple deployments where all the peers need to be grouped into a single IKEv2 profile irrespective of the peers' IKE identity. However, when the peers need to be grouped based on IKE identity, as in the case of managed service providers or enterprises with multiple departments, separate IKEv2 profiles must be configured with unique match statements.

The following example shows how to configure a profile that can match any peer identity.

Click here to view code image

```
Router(config-ikev2-profile)#match identity remote any ?
  <cr>

Router(config-ikev2-profile)#match identity remote any
Router(config-ikev2-profile)#match certificate certificate_map
%Error! please remove 'match identity remote any' statement
Router(config-ikev2-profile)#match identity remote address 192.168.1.1
%Error! please unconfigure 'match identity remote any' in this profile

Router(config-ikev2-profile)#no match identity remote any
Router(config-ikev2-profile)#match identity remote address 192.168.1.1
Router(config-ikev2-profile)#match identity remote any
%Error! please remove 'match certificate' and 'match identity remote' statements
Router(config-ikev2-profile)#
```

Defining the Scope of IKEv2 Profile

The scope of the IKEv2 profile can be defined using an optional **match fvrf** statement and one or more optional **match address local** statements that specify the FVRF and local address context of the IKEv2 negotiations for which the profile can be used. The default scope is global routing table and any local address.

The following example shows configuring FVRF and local address to define the scope of a profile.

Click here to view code image

```
Router(config-ikev2-profile)#match fvrf ?
  WORD  fvrf name (default: global)
  any   any fvrf

Router(config-ikev2-profile)#match address local ?
  A.B.C.D     Specify an IPv4 address
  X:X:X:X::X  Specify an IPv6 address
```

```
  interface   Specify an interface
```

Defining the Local IKE Identity

The IKEv2 profile allows defining the local IKE identity that is used to authenticate to the peers associated with the profile. The identity is carried in the identification payload (IDi/IDr) and is included in the calculation of the authentication payload in the IKE_AUTH messages to prove the identity to the peer. The local IKE identity is configured with the **identity local** statement. When it is not configured, the IKE negotiation local address is used as the local identity. Note that for an identity of type DN, the value is derived from the certificate, while for other identity types the value must be configured.

Table 5-11 shows the local IKEv2 Identity types supported in IKEv2 profile and their values.

Local Identity Type	Description
Address	IPv4 or IPv6 address
DN	Distinguished Name obtained from certificate
Email	RFC822 Email address
FQDN	A fully qualified domain name string
Key-id	Opaque string used for proprietary identification

Table 5-11 *IKEv2 Local Identity Types Supported in IKEv2 Profile*

The following example shows how to configure a local IKEv2 identity in an IKEv2 profile and display the identities used in an IKEv2 negotiation.

Click here to view code image

```
Router(config-ikev2-profile)#identity local address ?
  A.B.C.D     IPv4 address
  X:X:X:X::X  IPv6 address

Router(config-ikev2-profile)#identity local dn ?
  <cr>

Router(config-ikev2-profile)#identity local email ?
  WORD  email <user@domain>

Router(config-ikev2-profile)#identity local fqdn ?
  WORD  FQDN <host.domain>

Router(config-ikev2-profile)#identity local key-id ?
  WORD  key-id string
```

The IKE identities used by a initiator and responder in an IKEv2 SA can be seen in the highlighted portion of the output from **show crypto ikev2 sa detailed** and **show crypto ikev2 session detailed**.

Click here to view code image

```
Router#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         192.168.1.1/500           192.168.1.2/500           none/none
READY
```

```
      Encr: AES-CBC, keysize: 256, PRF: SHA512, Hash: SHA512, DH Grp:5, Auth sign:
PSK, Auth verify: PSK
      Life/Active Time: 86400/13 sec
      CE id: 1040, Session-id: 44
      Status Description: Negotiation done
      Local spi: 1935BB7ED97A8119       Remote spi: 4EB9F7349F07B7A4
      Local id: 192.168.1.1
      Remote id: 192.168.1.2
```

The identities exchanged between peers during the negotiation can be seen in the **debug crypto ikev2** logs.

On the initiator:

Click here to view code image

```
IKEv2:(SESSION ID = 3077,SA ID = 1):Constructing IDi payload: '192.168.1.1' of
type 'IPv4 address'
IKEv2:(SESSION ID = 3077,SA ID = 1):Searching policy based on peer's identity
'192.168.1.2' of type 'IPv4 address'
```

On the responder:

Click here to view code image

```
IKEv2:(SESSION ID = 1174,SA ID = 1):Searching policy based on peer's identity
'192.168.1.1' of type 'IPv4 address'
IKEv2:(SESSION ID = 1174,SA ID = 1):Constructing IDr payload: '192.168.1.2' of
type 'IPv4 address'
```

A local identity of type DN must be used only when the device is configured to authenticate itself using a certificate (that is, the local authentication method is certificate based) and a certificate is available. If not, the following error log is displayed when **debug crypto ikev2** is enabled, and the negotiation is aborted.

Click here to view code image

```
IKEv2-ERROR:(SESSION ID = 3077,SA ID = 1):: Detected an unsupported ID type
IKEv2:(SESSION ID = 3077,SA ID = 1):Auth exchange failed
IKEv2:(SESSION ID = 3077,SA ID = 1):Abort exchange
IKEv2:(SESSION ID = 3077,SA ID = 1):Deleting SA
```

Defining Local and Remote Authentication Methods

Unlike IKEv1, IKEv2 does not negotiate the authentication method between peers and uses unidirectional authentication, which means the method used by a device to authenticate to its peer is independent of the method used by the peer to authenticate to the device.

The IKEv2 profile defines local and remote authentication methods to support IKEv2 unidirectional authentication. The local authentication method is used by the device to authenticate to its peers, and the remote authentication method is used by the device to authenticate or validate the peers. While the local and remote authentication methods are independent in general, when the local authentication method is EAP based, the remote authentication method must be certificate-based and vice versa. Note that the local authentication method configured on a device must be configured as a remote authentication method on the peer and vice versa.

Only one local authentication method can be configured, while multiple remote authentication methods can be configured; this allows an IKEv2 profile to serve peers authenticating using different

methods.

The authentication method used by a device is denoted in the authentication payload in the IKE_AUTH exchange; this allows the peer to select the appropriate method to validate the device. The following packet capture shows the authentication method in the authentication payload.

Click here to view code image

```
Type Payload: Authentication (39)
                    Next payload: Security Association (33)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 28
                    Authentication Method: Shared Key Message Integrity Code (2)
                    Authentication Data: 27c5615a79ba2abe1c0077507770d624c3f8e246
```

The following local and remote authentication methods are supported.

Click here to view code image

```
Router(config-ikev2-profile)#authentication local ?
  eap         Extended Authentication Protocol
  ecdsa-sig   ECDSA Signature
  pre-share   Pre-Shared Key
  rsa-sig     Rivest-Shamir-Adleman Signature

Router(config-ikev2-profile)#authentication remote ?
  anyconnect-eap  AnyConnect EAP
  eap             Extended Authentication Protocol
  ecdsa-sig       ECDSA Signature
  pre-share       Pre-Shared Key
  rsa-sig         Rivest-Shamir-Adleman Signature
```

> **Note**
>
> The EAP and AnyConnect-EAP authentication methods are mainly used with FlexVPN Client and Server respectively; hence they are described in the respective chapters for those features. Unlike pre-shared key and certificate-based authentication methods that are used for device authentication, the EAP method is used for user authentication and is mainly used with FlexVPN client (Remote Access Client). It is in chapter 10, "FlexVPN Client." The AnyConnect-EAP method is Cisco proprietary. The AnyConnect client-specific method that is supported by FlexVPN Server is described in chapter 9, "FlexVPN Server."

Pre-Shared Key Authentication

When the pre-shared key method is used for local or remote authentication, the pre-shared keys can be configured in the IKEv2 profile and/or in a local or remote (RADIUS-based) keyring with the former taking precedence.

Below is an example of pre-shared key authentication with keys directly configured in the profile as part of the **authentication** command. Note that the keys configured in the profile take precedence and any keyring configuration in the profile is ignored. A limitation of this approach is that all peers matching this profile must use the same local and remote key, while with local or RADIUS-based keyring, the keys can be per peer. This is useful is simple deployments that use the same pre-shared key for all the peers associated with an IKEv2 profile.

Click here to view code image

```
crypto ikev2 profile ikev2_profile
 authentication local pre-share key key1
 authentication remote pre-share key key2
```

The following example shows configuring pre-shared key authentication with keys configured in a local keyring that is referenced from the profile. Note that keys can be configured on a per-peer or peer-group basis.

Click here to view code image

```
crypto ikev2 keyring ikev2_keyring
 peer peer1
  address 192.168.2.1 255.255.255.255
  pre-shared-key local key1 remote key2

 peer peer2
  address 192.168.3.0 255.255.255.0
  pre-shared-key local key3 remote key4

crypto ikev2 profile ikev2_profile
 keyring local ikev2_keyring
```

> **Note**
>
> RADIUS-based pre-shared keys are mainly used by FlexVPN Server for scalable and centralized management of pre-shared keys; they are described in the FlexVPN Server chapter

Certificate-Based Authentication

Certificate-based authentication is generally considered cryptographically more secure and scalable than the pre-shared key method. It uses a Public Key Infrastructure (PKI) to manage the issuing of certificates, revocation of certificates, protection of key material, and security policy. With this method, a device proves its identity by signing a known blob of data using its private key, and the peer verifies the same using the corresponding public key conveyed in the certificate. The authentication data and the certificate are carried in the authentication (AUTH) and certificate (CERT) payloads in IKE_AUTH exchange, and the peer indicates the certificate authorities that are acceptable in the CERTREQ payload as shown below.

Click here to view code image

```
Initiator                                          Responder
---------                                          ---------
IKE_SA_INIT Request:
HDR, SAi1, KEi, Ni  -->
                                        IKE_SA_INIT Response:
                                   <--  HDR, SAr1, KEr, Nr, [CERTREQ]
IKE_AUTH Request:
HDR, SK {IDi, [CERT,] [CERTREQ,]
[IDr,] AUTH, SAi2, TSi, TSr}  -->
                                      IKE_AUTH Response:
                                   <--  HDR, SK {IDr, [CERT,] AUTH,
                                        SAr2, TSi, TSr}
```

The PKI trustpoint configuration specifies the CA server and related parameters for certificate enrollment and validation. Note that the term trustpoint implies the corresponding CA server in the description below.

When a certificate-based method (RSA-SIG or ECDSA-SIG) is used for local or remote authentication, a PKI trustpoint must be configured and referenced from the IKEv2 profile. Note that only the trustpoints configured in the IKEv2 profile are used for signing and validation and must be configured.

If a certificate-based method is used for remote authentication, a PKI trustpoint must be configured, and the CA certificate must be obtained from the CA server in order to validate the peer certificate. If the certificate method is used for local authentication, then in addition to the PKI trustpoint configuration and obtaining the CA server certificate, an RSA/EC key pair must be generated, and the device certificate must be obtained from the CA server. Please refer to the "Configuring PKI in Cisco IOS" section of this chapter for a detailed description of CA Server and PKI trustpoint configuration, generation of the PKI key pair, and ways of obtaining CA server and device certificates.

Up to six trustpoints can be configured in an IKEv2 profile that can be used for signing or validation or both. Note that a trustpoint configured without the **sign** or **verify** keyword can be used for both signing and verification.

Click here to view code image

```
Router(config-ikev2-profile)#pki trustpoint ca_server1 ?
  sign    Set trustpoint for sign
  verify  Set trustpoint for verify
  <cr>
```

Table 5-12 summarizes the trustpoint (TP) selection for sending a certificate request (CERTREQ) and certificate (CERT) payloads and for certificate validation on the initiator and responder.

	Trustpoint Selection for:		
	Sending CERTREQ	**Sending CERT**	**CERT Validation**
Initiator	Sent in IKE_AUTH request; TPs are selected from IKEv2 profile. No fallback to globally configured TPs if no TPs are found in IKEv2 profile.	TP is selected from IKEv2 profile that matches the TP received in the CERTREQ. No fallback to globally configured TPs for sending CERT.	TP is selected from IKEv2 profile that matches the received CERT's TP. No fallback to globally configured TPs for CERT validation.
Responder	Sent in IKE_SA_INIT response; IKEv2 profile selection is not done yet, so the globally configured TPs are used.	TP is selected from IKEv2 profile that matches the TP received in the CERTREQ. No fallback to globally configured TPs for sending CERT.	TP is selected from IKEv2 profile that matches the received CERT's TP. No fallback to globally configured TPs for CERT validation.

Table 5-12 *Trust Point Selection for CERTREQ, CERT, and CERT Validation*

The following example shows how to configure certificate authentication and a PKI trustpoint.

Click here to view code image

```
crypto pki trustpoint ca_server1
 enrollment url http://192.168.1.4:80
 subject-name cn=Router
 revocation-check crl

crypto ikev2 profile ikev2_profile
 match identity remote any
 authentication local rsa-sig
 authentication remote rsa-sig
 pki trustpoint ca_server1
```

IKEv2 Dead Peer Detection

Dead Peer Detection (DPD) is a mechanism whereby a device will send a liveness check to its IKEv2 peer to check that the peer is functioning correctly. These checks are sent from each peer as an empty INFORMATIONAL exchange, which the corresponding receiving peer receives and retransmits back to the initiating peer. The peer that initiated the liveness check can validate the returned packet by noting the message ID.

If a device sends a liveness check to its peer and fails to receive a response, then the device will go into an aggressive retransmit mode, in which it will transmit five DPD messages at a configured interval. If these transmitted DPD exchanges are not acknowledged, then the peer device will be marked as dead, and the IKEv2 SA and the child IPsec Security Associations will be torn down. The following example shows how to configure Dead Peer Detection (DPD) in an IKEv2 profile.

Click here to view code image

```
Router(config-ikev2-profile)#dpd ?
  <10-3600>  DPD R-U-THERE interval

Router(config-ikev2-profile)#dpd 10 ?
  <2-60>  DPD Retry Interval

Router(config-ikev2-profile)#dpd 10 2 ?
  on-demand  Send DPD queries only as needed
  periodic   Send DPD queries at regular intervals
```

Note that DPD can also be configured globally in the router configuration mode. The IKEv2 profile DPD configuration overrides the global DPD configuration for the peers or IKEv2 sessions associated with that profile.

Periodic DPD forces DPD messages to be sent at regular intervals. A message is sent at every configured time interval, irrespective of any traffic received on the IPsec Security Association relating to the IKEv2 SA.

On-demand mode will monitor received traffic on the IPsec Security Association that was built from the IKEv2 SA only when needed, that is, before sending any outbound traffic on the IPsec Security Association. If no received traffic is seen on this IPsec Security Association for the configured period, the host will send a DPD message to its peer to ascertain if it is alive. If a response is not received, the device will go into an aggressive mode and retransmit DPD messages every interval. If any replies to the DPD messages are received or if traffic is seen on the corresponding IPsec Security Associations, it will exit from the aggressive mode; however, if no replies are received after five aggressive attempts, the IKEv2 SA and the child IPsec Security Associations will be torn down.

Periodic mode can result in earlier detection of dead peers than on-demand mode. For example, if a peer lost connectivity and a DPD message was sent by the router just after this occurred, after no reply to the DPD message the aggressive mode would be enabled. However, with on-demand mode the DPD messages would be sent only if there is any outbound traffic to be sent and no traffic has been received for the configured duration, and in the absence of outbound traffic, on-demand DPD may not detect if the peer is dead. When on-demand mode is used, if an IKEv2 SA has more than just a single pair of IPsec Security Associations, traffic received on any of the IPsec Security Associations will disable the sending of DPD messages.

When hub and spoke networks are designed in which many spoke devices will connect to a VPN headend, care should be taken if DPD messages are enabled that originate from the hub to the spokes. DPD messages consume CPU cycles, and should a large number of spokes suddenly lose connectivity to the hub, then the hub CPU could become hogged in generating a large amount of DPD packets. It is advised that for large hub and spoke topologies only the spoke should have DPD enabled, and should connectivity fail between the hub and the spoke, then the spoke will transition into the aggressive retransmit state to detect failed connectivity and tear down the IKEv2 and child SAs. When connectivity is restored between the spoke and the hub, the newly created IKEv2 Security Association on the hub will tear the old security association down due to INITIAL_CONTACT.

There is one caveat with not enabling DPD messages on hub devices. Should connectivity be lost, then the hub will continue to send encrypted traffic to a spoke that will not receive the data until the spoke establishes a new IKEv2 SA or until a rekey occurs and is unsuccessful, resulting in the deletion of the SA.

IKEv2 Initial Contact

Should an IKEv2 device crash or be reloaded and DPD is not enabled on the peer device, then the IKEv2 SA and all corresponding IPsec Security Associations would become stale and eventually time out. When the crashed device initiates a new IKEv2 session, it would use the same IKEv2 identity, so the peer with the stale IKEv2 SA would realize then that this is a duplicate session and delete the old stale session. The behavior to check the identity of previously created IKEv2 sessions and delete any existing sessions is enabled when a peer sends an INITIAL_CONTACT notification payload within the IKE_AUTH exchange. This will ensure that any previously created sessions are deleted.

This behavior is enabled by default on Cisco IOS; however, should an IKEv2 implementation initiate an IKEv2 session that does not contain the INITIAL_CONTACT notification payload, then the initial contact processing based on identity and 5 tuple session information for any existing stale IKEv2 session with that peer can be forced by using the following command in IKEv2 profile.

Click here to view code image

```
crypto ikev2 profile ikev2_profile
  initial-contact force
```

IKEv2 SA Lifetime

IKEv2 SAs use secret keys that should be used only for a limited amount of time. This limits the lifetime of the entire SA. When the lifetime of an SA expires, the SA must not be used and must be replaced with new SAs that use new keys. In IKEv2, the SA lifetime is not negotiated, and each end of the SA is responsible for enforcing its own lifetime policy on the SA and rekeying the SA when necessary. This allows better control over which side initiates the rekey and is useful in spoke-hub topologies, where the spokes can be configured with a lower lifetime and made to initiate the rekeys instead of the hub. The peer with the shorter lifetime will always end up being the one to request the rekeying. Cisco IOS uses a default IKEv2 SA lifetime of 24 hours, which can be overridden on each IKEv2 profile with the **lifetime** *lifetime_duration* command, where the lifetime can be configured between 120 and 86,400 seconds.

NAT Keepalives

NAT detection and NAT traversal is built into the IKEv2 protocol. When the use of NAT is detected, as part of NAT traversal, IKEv2 moves from UDP port from 500 to 4500. Additionally, IKEv2 programs the child IPsec Encapsulation Security Payload SAs to use UDP encapsulation, the peer's post-NAT IP address and UDP port number, and the frequency at which the NAT keepalives must be sent to keep the NAT translation entries alive on the NAT device. Please refer to RFC3948 (UDP Encapsulation of IPsec Encapsulation Security Payload Packets) for a description of the NAT keepalive packet format and procedure. The following example shows how to configure the NAT keepalive frequency in the IKEv2 profile.

Click here to view code image

```
Router(config-ikev2-profile)#nat ?
  keepalive  NAT keepalive interval

Router(config-ikev2-profile)#nat keepalive ?
  <5-3600>  NAT keepalive interval in seconds
```

Note that NAT keepalive frequency can also be configured globally in the router configuration mode with **crypto ikev2 nat keepalive command** and, as always, the IKEv2 profile configuration overrides the global configuration for the peers or IKEv2 sessions associated with that profile.

IVRF (inside VRF)

As described earlier in chapter 4, "IOS IPsec implementation," VRF is a Cisco IOS layer-3 device virtualization technique used to segregate layer-3 forwarding domains. IOS uses two types of VRFs: IVRF (inside) and FVRF (front-door VRF). In IOS, IVRF is used for segregation of overlay domains to support multi-tenancy, and FVRF is used for segregation of underlay domains (WAN transports/providers). As IPsec deals with both overlay (plaintext) and underlay (encrypted) traffic, IPsec Security Associations use both IVRF and FVRF. With tunnel protection, IPsec Security Associations derive FVRF and IVRF from the tunnel interface, which can be configured with both VRFs. With crypto maps, IPsec Security Associations derive FVRF from the crypto map interface and IVRF from the IKEv2 profile used by the SAs as crypto map interfaces, which are typically non-tunnel interfaces can be configured only with a single VRF. The following example shows configuring VRF in the IKEv2 profile.

The IVRF can be configured in IKEv2 profile with the **ivrf** command, as shown below:

Click here to view code image

```
Router(config-ikev2-profile)#ivrf ?
  WORD VRF Name
```

Note that IVRF configuration in IKEv2 profile is meant to be used only with crypto maps and must not be used with tunnel protection.

Virtual-Template Interface

The IPsec dVTI is a dynamically instantiated virtual access interface of type tunnel that is used on IPsec VPN headends such as FlexVPN server. A dVTI is created for every incoming IKE and IPsec session, and the configuration for a dVTI is primarily cloned from a virtual-template interface of type tunnel and optionally from AAA. The virtual-template used for an incoming session is derived from the IKEv2 profile associated with that session. For a dVTI, a virtual-template must be configured in an IKEv2 profile, without which the virtual access interface is not created. The following example shows configuring a virtual-template and referencing it from an IKEv2 profile.

Click here to view code image

```
Router(config-ikev2-profile)#?
  virtual-template  Specify the virtual-template for dynamic interface
                    creation.

Router(config-ikev2-profile)#virtual-template ?
  <1-200>  Virtual-template number


interface Virtual-Template1 type tunnel
 ip unnumbered Ethernet0/0
 tunnel source Ethernet0/0
 tunnel mode ipsec ipv4
 tunnel protection ipsec profile ipsec_profile
```

```
crypto ikev2 profile ikev2_profile
virtual-template 1
```

Disabling IKEv2 Profile

It is recommended that an IKEv2 profile be disabled when it is being configured or modified; this prevents an incomplete profile from being used, which would cause undesired results. A profile can be disabled with the **shutdown** command.

Click here to view code image

```
Router(config-ikev2-profile)#shutdown ?
  <cr>
```

Displaying IKEv2 Profiles

The configured IKEv2 profiles can be displayed with the following command. The output also displays default values used in the IKEv2 profile:

Click here to view code image

```
Router#show crypto ikev2 profile

IKEv2 profile: default
 Ref Count: 5
 Match criteria:
  Fvrf: global
  Local address/interface: none
  Identities: any
  Certificate maps: none
 Local identity: none
 Remote identity: none
 Local authentication method: pre-share
 Remote authentication method(s): pre-share
 EAP options: none
 Keyring: ikev2_keyring
 Trustpoint(s): none
 Lifetime: 86400 seconds
 DPD: disabled
 NAT-keepalive: disabled
 Ivrf: none
 Virtual-template: none
 mode auto: none
 AAA AnyConnect EAP authentication mlist: none
 AAA EAP authentication mlist: none
 AAA Accounting: none
 AAA group authorization: none
 AAA user authorization: none
```

IKEv2 Profile Selection on Initiator and Responder

On the initiator, the IKEv2 profile used for the outbound negotiation is the one that is attached to the IPsec interface (for example, the tunnel interface with tunnel protection applied on it) that initiated the IPsec Security Association request. The profile is later validated against the peer certificate or identity received in the IKE_AUTH response, and if there is a mismatch, the negotiation is aborted.

On the responder, the IKEv2 profile is selected based on the peer certificate or identity received in the IKE_AUTH request, and this profile is later validated against the profile attached to the IPsec interface that accepted the incoming the IPsec session. This check is performed during IPsec proposal

validation, and if there is a mismatch the negotiation is aborted.

IKEv2 Profile Key Points

Following is a summary of the key points about IKEv2 profiles.

- The IKEv2 profile is a mandatory configuration on the initiator and responder, for it specifies key parameters, such as authentication methods and credentials, without which negotiation cannot start.
- The IKEv2 profile must be attached to an IPsec interface on both the initiator and responder that defines the IKE identities the interface can communicate with.
- The IKEv2 profile attached to the IPsec interface must match the profile selected during the negotiation based on peer certificate or identity. If not, the negotiation is aborted.
- The IKEv2 profile configuration overrides any corresponding global configuration and applies only to peers or sessions that get associated with that profile.

IKEv2 Global Configuration

The IKEv2 global configuration parameters defined in the router configuration mode apply to all the IKEv2 sessions and peers, irrespective of the IKEv2 profile that they associated with.

Table 5-13 lists the IKEv2 parameters that can be configured globally and the functionality that they provide.

IKEv2 Global Parameter	Functionality Provided
http url	Certificate lookup from an http url
Certificate cache	Caching of certificates fetched from http url
Cookie challenge	Threshold for using anti-DOS cookie challenge
SA limit	IKEv2 call admission control
Window size	Number of simultaneous IKEv2 requests
Dead peer detection	DPD method and intervals
NAT keepalive	NAT keepalive frequency
CTS	Cisco Trustsec SGT capability negotiation
Disconnect revoked peers	Delete session when peer certificate is revoked
Fragmentation	Fragmentation of IKEv2 packets
Redirect	IKEv2 redirect-based server load balancing
Reconnect	Automatic reconnection of IKEv2 sessions

Table 5-13 *IKEv2 Global Configuration Parameters*

Note

IKEv2 fragmentation, CTS, and the session deletion on peer certificate revocation feature are described in chapter 6, "Advanced IKEv2 Features." The IKEv2 redirect and auto-reconnect features are described in chapter 11, "FlexVPN Load Balancer," and chapter 9, "FlexVPN Server."

HTTP URL-based Certificate Lookup

If the IKEv2 messages are long enough that IP-level fragmentation is necessary, it is possible that malicious users could prevent the exchange from completing by exhausting the reassembly buffers. The chances of this can be minimized by using the Hash and URL encoding instead of sending certificates where the peer would fetch the certificate using http from the specified URL.

The HTTP_CERT_LOOKUP_SUPPORTED notification as part of the CERTREQ payload indicates that the sender is capable of looking up certificates based on an HTTP-based URL and would prefer to receive certificate specifications in that format. If the peer supports that feature, the certificate payload includes a hash and URL of the certificate instead of the certificate. The following example shows how to enable HTTP URL-based certificate lookup.

Click here to view code image

```
Router(config)#crypto ikev2 http-url ?
  cert Enable certificate lookup based on HTTP-based URL
```

The caching of certificates fetched from HTTP URL for future use is enabled with the following command:

Click here to view code image

```
Router(config)#crypto ikev2 ?
  certificate-cache     Cache for storing certs fetched from HTTP URLs

Router(config)#crypto ikev2 certificate-cache ?
  <1-2000>  Maximum number of certificates stored in certificate cache
```

IKEv2 Cookie Challenge

IKEv2 Cookie Challenge is used to mitigate against a possible DoS attack on an IKEv2 responder that results from a flood of session initiation requests from forged IP addresses. The mitigation involves using minimal CPU and state resources until the responder confirms that the initiator can receive packets at the address from which it claims to be sending them. When a responder detects a large number of half-open IKE SAs, it replies to IKE_SA_INIT requests with a response containing the COOKIE notification with a stateless cookie, and only when the initiator reflects the cookie back does it proceed with the negotiation. The following example shows configuring the threshold for the number of half open IKEv2 session requests, after which cookie challenge is enabled.

Click here to view code image

```
Router(config)#crypto ikev2 ?
  cookie-challenge     Set Cookie-challenge watermark

Router(config)#crypto ikev2 cookie-challenge ?
  <0-1000>  Maximum number of in-negotiation SAs that enable Cookie-challenge
```

IKEv2 Call Admission Control

IKEv2 Call Admission Control (CAC) limits the maximum number of IKEv2 SAs that can be established by a device based on the platform limitations, such as crypto engine or CPU limitations. CAC also limits the number of simultaneous negotiations, depending on the device capabilities, to avoid thrashing because setting up a new IKEv2 session is resource intensive. The limits will vary depending on the platform, and tuning the limits based on testing will yield the optimal performance and session setup rate. At the time of this writing, the default IKEv2 CAC values are 40 for incoming sessions and 400 for outgoing sessions.

The following example shows how to configure the CAC limits.

Click here to view code image

```
Router(config)#crypto ikev2 ?
  limit                 Limit the number of maximum and negotiating sa

Router(config)#crypto ikev2 limit ?
  max-in-negotation-sa  Limit the maximum number of negotiating SA
  max-sa                Limit the maximum number of SA allowed

Router(config)#crypto ikev2 limit max-sa ?
  <1-9999>  Set the limit

Router(config)#crypto ikev2 limit max-in-negotation-sa ?
  <1-9999>  Set the limit

Router(config)#crypto ikev2 limit max-in-negotation-sa 100 ?
  incoming  Limit the maximum number of incoming negotiating SA
  outgoing  Limit the maximum number of outgoing negotiating SA
  <cr>
```

The IKEv2 call admission statistics can be seen with the **show crypto ikev2 stats** command:

Click here to view code image

```
Router#show crypto ikev2 stats
------------------------------------------------------------------------------
                          Crypto IKEv2 SA Statistics
------------------------------------------------------------------------------
System Resource Limit:   0        Max IKEv2 SAs: 0        Max in nego(in/out):
100/400
Total incoming IKEv2 SA Count:    0        active:        0        negotiating: 0
Total outgoing IKEv2 SA Count:    1        active:        1        negotiating: 0
Incoming IKEv2 Requests: 0        accepted:      0        rejected:    0
Outgoing IKEv2 Requests: 3        accepted:      3        rejected:    0
Rejected IKEv2 Requests: 0        rsrc low:      0        SA limit:    0
IKEv2 packets dropped at dispatch: 0
Incoming IKEV2 Cookie Challenged Requests: 0
    accepted: 0        rejected: 0        rejected no cookie: 0
Total Deleted sessions of Cert Revoked Peers: 0
```

IKEv2 Window Size

Each IKEv2 header contains a message identifier, which is a sequential number beginning at 0, for every exchange this value is incremented by one for every pair of exchanged packets. Having a message number within the header allows each packet exchange to be tracked and prevents replay attacks.

All IKEv2 exchanges are in pairs, in which a sent packet will always be acknowledged by the receiver. The number of pairs of messages that can be inflight at any time is exchanged by using the SET_WINDOW_SIZE notification payload within the IKE_SA_INIT exchange. This value denotes the IKEv2 window size, and by default Cisco IOS sets the window size to 5, which allows up to five IKEv2 exchanges to be sent before an acknowledgement is received. There are very few cases in which more than five packets will be inflight at any given time. If dead peer detection is enabled and INFORMATIONAL exchanges occur around the same time that a rekey occurs, then two or three packet exchanges could be occurring at the same time, but the default window size of 5 should be ample for this scenario.

On IOS the SET_WINDOW_SIZE notification payload is sent in the IKE_AUTH exchange; once the IKEv2 Window Size has been set it cannot be reduced but can only grow in size.

Click here to view code image

```
    Type Payload: Notify (41)
                  Next payload: Notify (41)
                  0... .... = Critical Bit: Not Critical
                  Payload length: 12
                  Protocol ID: IKE (1)
                  SPI Size: 0
                  Notify Message Type: SET_WINDOW_SIZE (16385)
                  Notification DATA: 00000005

IKEv2 window size can be configured globally using the following
command:Router(config)#crypto ikev2 window ?
  <1-200>  Maximum number of overlapping requests with a peer
```

Dead Peer Detection

Please refer to the IKEv2 profile section for a detailed description of Dead Peer Detection (DPD). The DPD method and intervals can be configured globally with the following command:

Click here to view code image

```
Router(config)#crypto ikev2 dpd ?
  <10-3600>  DPD R-U-THERE interval

Router(config)#crypto ikev2 dpd 10 ?
  <2-60>  DPD Retry Interval

Router(config)#crypto ikev2 dpd 10 2 ?
  on-demand  Send DPD queries only as needed
  periodic   Send DPD queries at regular intervals
```

NAT Keepalive

Please refer to the IKEv2 profile section for a detailed description of NAT keepalives. NAT keepalives can be configured globally with the following command:

Click here to view code image

```
Router(config)#crypto ikev2 nat keepalive ?
  <5-3600>  Number of seconds between NAT keepalive
```

IKEv2 Diagnostics

IKEv2 diagnostics maintains a database of exit paths and a count of different error types. The exit path captures the exit points from the normal code paths caused by errors, and the accompanying tracebacks help determine the exact location of the error, because the same error message can be generated from multiple places in the code.

IKEv2 diagnostics can be enabled with the following command:

Click here to view code image

```
Router(config)#crypto ikev2 diagnose error ?
  <1-1000> Maximum number of exit path entry
```

The exit path database and the error counts can be displayed with the following command:

Click here to view code image

```
Router#show crypto ikev2 diagnose ?
  error  Shows error trace database

Router#show crypto ikev2 diagnose error ?
  count  Shows ikev2 errors counts - only non-zero
  |      Output modifiers
  <cr>

Router#show crypto ikev2 diagnose error count ?
  detailed  Shows ikev2 errors counts - all
  |         Output modifiers
  <cr>
```

PKI Configuration

Public Key Infrastructure (PKI) is a very large and complicated topic; indeed, a whole book could easily be written to describe how PKI can be implemented on Cisco IOS. The following title by Cisco Press is that book and should be read for an in-depth study of PKI: *PKI Uncovered: Certificate-Based Security Solutions for Next-Generation Networks*.

Within this section we cover the core components of PKI and then use an example to illustrate the component in action.

A number of components are found within a PKI. On Cisco IOS, the relevant ones are the certificate authority, public-private key pair, and certificate.

Certificate Authority

A Certificate Authority (CA) manages certificate requests and issues certificates to devices participating within the PKI. These services (managing certificate requests and issuing certificates) provide centralized key management for the participating devices and are explicitly trusted by the receiver to validate identities and to create digital certificates.

Cisco IOS can operate as a CA using a Rivest-Shamir-Adleman (RSA) key pair. This can be very useful in case a single IPsec Security Association is required to be established between a pair of routers and a fully blown PKI is not required. One of the routers can act as a CA and the other router can enroll into this. The following command can be used to configure a Cisco IOS router as a CA.

```
crypto pki server cs-label
```

where *cs-label* is the name of the CA server.

After configuring the CA function, a submenu will be presented in which additional configuration options can be applied to specific attributes pertaining to the CA. The following examples shows CA server configuration.

Click here to view code image

```
Router(config)#crypto pki server CA
Router(cs-server)#?
CA Server configuration commands:
  auto-rollover  Rollover the CA key and certificate
  cdp-url        CRL Distribution Point to be included in the issued certs
  crl            server crl
  database       Certificate Server database config parameters
  default        Set a command to its defaults
  eku            Configure EKU parameters
  exit           Exit from Certificate Server entry mode
  grant          Certificate granting options
  hash           Hash algorithm
  issuer-name    Issuer name
  lifetime       Lifetime parameters
  mode           Mode
  no             Negate a command or set its defaults
  redundancy     sync this server to the standby
  serial-number  Serial Number of Last Certificate Issued
  show           Show this certificate server configuration
  shutdown       Shutdown the Certificate Server
```

To activate the CA service, issue the **no shutdown** command within the CA server. The CA server will become active when the HTTP service is enabled locally. The following example illustrates a method to determine if the CA server is disabled due to the HTPP service not being enabled.

Click here to view code image

```
Router#show crypto pki server
Certificate Server CA:
    Status: disabled, HTTP Server is disabled
```

Enter the **ip http server** global command to activate the HTTP service. Access to the HTTP service should be restricted by using the **ip http access-class** ***access-list-number*** global command.

A password is required, which is used to protect the private key that is used by the CA service.

Click here to view code image

```
Router(config)#crypto pki server CA
Router(cs-server)#no shut
%Some server settings cannot be changed after CA certificate generation.
% Please enter a passphrase to protect the private key
% or type Return to exit
Password:

Re-enter password:
% Generating 1024 bit RSA keys, keys will be non-exportable...
[OK] (elapsed time was 0 seconds)
```

When the CA is created, an RSA key pair will be generated if one is not defined. This will be a default 1024-bit RSA key pair. For maximum security, a stronger key pair should be used, which is described in the following section.

The following output illustrates the expected behavior once the CA service is enabled.

Click here to view code image

```
%PKI-6-CS_ENABLED: Certificate server now enabled.
Router#show crypto pki server
Certificate Server CA:
    Status: enabled
    State: enabled

....
```

Accurate time is crucial when designing the architecture of a Certificate Authority; for this reason, Network Time Protocol (NTP) should always be used to ensure accurate time when a PKI is used.

The example above is a simplified PKI deployment. In cases where a subordinate CA is used, multiple trustpoints would be required if full chain validation is needed.

Public-Private Key Pair

A key pair consists of a public key and a private key. The two keys in the key pair have a cryptographic relationship with each other, so that what can be done with one can be undone with the other and vice versa. When a PKI is set up, the public key is required in the certificate enrollment request. Once the certificate has been granted, the public key will be included in the certificate. In the context of IKEv2, the public key is used to verify that the peer is in possession of the private key.

> **Note**
>
> Prior to generating the default RSA key pair the router hostname and domain-name must be set. This requirement does not exist for named key pairs.

To generate a key pair, use the **crypto key generate** command. This command can generate either a Rivest-Shamir-Adleman (RSA) or Elliptic Curve (EC) key pair. EC keys pairs are smaller and offer greater security for less work than RSA; however, they have only recently been introduced and many third-party devices do not support EC key pairs. The following example shows EC key pair generation.

Click here to view code image

```
crypto key generate ec keysize { 256 | 384 } [ exportable ] [ label key-label ]
```

The following global command is used to generate an RSA key pair.

Click here to view code image

```
crypto key generate rsa [ general-keys | usage-keys | signature | encryption ]
[ label key-label ] [exportable] [ modulus modulus-size ] [ storage devicename : ]
[redundancy] [ on devicename : ]
```

A Cisco IOS router can be enrolled into a PKI. Before enrollment, a public-private key pair must be available on the router. The generation of the public-private key pair can be performed locally with the **crypto key generate** command, or the key pair can be imported.

The following command illustrates how an EC key pair can be imported.

Click here to view code image

```
crypto key import ec key-label [ exportable ] { terminal | url url } passphrase
```

The following command illustrates how a RSA key pair can be imported.

Click here to view code image

```
crypto key import rsa key-label pem [ usage-keys | signature | encryption |
general-purpose ] { storage | terminal [passphrase] | url url } [exportable]
[ on devicename : ]
```

A public-private key pair can be removed with the following commands:

Click here to view code image

```
crypto key zeroize ec [key-pair-label]

crypto key zeroize rsa [key-pair-label]
```

PKI Trustpoint

A PKI trustpoint is a container for information pertaining to a certificate authority. The trustpoint will contain the certificate of the CA in question, along with attributes that are used by the local device when generating a Certificate Signing Request (CSR).

The sending of the CSR to the CA can be performed by a manual process (copy and paste) which uses the **enrollment terminal** command from within the trustpoint. Simple Certificate Enrolment Protocol (SCEP) is an HTTP-based method to enroll clients to a CA with RSA certificates. SCEP can be enabled with the **enrollment url** *scep-url* command, where *url* is the URL of the CA to which the router should send certificate requests.

If EC certificates are used, Enrollment over Secure Transport (EST) can be used to enroll next-generation encryption certificates over SSL or TLS. EST is described with RFC7030. To configure enrollment using EST, a PKI profile is required and created with the command **crypto pki profile enrollment** *pki_profile*. Within this profile, the command **method-est** is used to enable enrolment using EST. The trustpoint that is configured to use EST must be defined using the command **enrollment credential** *trustpoint_label*. The enrollment URL must be defined using the command **enrollment url** *url*.

The public-private key pair that is used for the CSR can be defined within the trustpoint. To configure an EC key pair, the **eckeypair** *name* command is used. To configure a RSA key pair the **rsakeypair** *key-label [[key-size encryption-key-size]]* command is used. The *key-size* argument is used for generating the key, and the *encryption-key-size* argument specifies that the system is to request separate encryption, signature keys, and certificates. The *key-size* and *encryption-key-size* values must be the same size. For RSA, a length of less than 2048 is not recommended.

To specify the cryptographic hash algorithm function for the signature that the routers uses for the CSR, the **hash** *hash_type* command is used. Table 5-14 lists the available hash algorithms and their cryptographic strength.

Hash Algorithm	Strength	QCR
Sha521	NGE	Yes
Sha384	NGE	Yes
Sha256	NGE	No
Sha1	Legacy	No
Md5	Avoid	No

Table 5-14 *Hash Algorithms for Signature*

The certificate revocation method is performed to ensure that the certificate has not been revoked by the issuing CA. Cisco IOS software supports two revocation mechanisms: Certificate Revocation Lists (CRLs) and Online Certificate Status Protocol (OCSP). A CRL is a list of revoked certificates. The CRL is created and digitally signed by the CA that originally issued the certificates. The CRL contains the serial number, date and revocation reason for each certificate that was revoked.

CAs publish new CRLs periodically or when a certificate for which the CA is responsible has been revoked. By default, a new CRL is downloaded after the currently cached CRL expires. An administrator may also configure the duration for which CRLs are cached in router memory or disable CRL caching completely. The CRL caching configuration applies to all CRLs associated with a trustpoint.

The ability to check if a peer's certificate has been revoked is achieved using the **revocation-check** *method* command.

A number of other attributes can be configured within the trustpoint, where the revocation method can be one or more of the following (please consult cisco.com for more information):

- crl—Certificate checking is performed by a certificate revocation list (CRL). This is the default behavior.
- none—Certificate checking is not required.
- ocsp—Certificate checking is performed by an online certificate status protocol (OCSP) server.

Once a trustpoint has been configured, the CA certificate must be obtained. This is achieved by authenticating the CA with the global command **crypto pki authenticate** *name*, where *name* is the name of the trustpoint. The process to obtain the CA certificate is achieved by cut and paste when enrollment is set to terminal, or by SCEP when an enrollment URL is defined.

The fingerprint of the CA certificate is its MD5 and SHA1 hash and can be used to verify that it was indeed the genuine CA certificate that was received.

Once the CA certificate is obtained, the router can be enrolled into the PKI with the global command **crypto pki enroll** *name*, where *name* is the name of the trustpoint.

In the enrollment process, a CSR that is populated with the details defined within the trustpoint is sent to the CA; the CA will cryptographically sign the hash of this CSR and return it to the client. This is the client's certificate.

PKI Example

The following example illustrates a router being configured to act as a Certificate Authority.

The host name and domain name are first set, then an RSA keypair used by the CA is generated. The CA will be configured to automatically generate certificates. For security purposes this should not be configured.

Click here to view code image

```
Router(config)#hostname CA
CA(config)#ip domain-name cisco.com
CA(config)#crypto key generate rsa general-keys modulus 2048 label CA
The name for the keys will be: CA

% The key modulus size is 2048 bits
% Generating 2048 bit RSA keys, keys will be non-exportable...
[OK] (elapsed time was 1 seconds)

CA(config)#crypto pki trustpoint CA
CA(ca-trustpoint)#rsakeypair CA
CA(ca-trustpoint)#exit
CA(config)#crypto pki server CA
CA(cs-server)#grant auto
%PKI-6-CS_GRANT_AUTO: All enrollment requests will be automatically granted.
CA(cs-server)#no shutdown
%Some server settings cannot be changed after CA certificate generation.
% Please enter a passphrase to protect the private key
% or type Return to exit
Password:

Re-enter password:

% Certificate Server enabled.
CA(cs-server)#
%PKI-6-CS_ENABLED: Certificate server now enabled.

CA#show crypto pki server
Certificate Server CA:
    Status: enabled
    State: enabled
    Server's configuration is locked  (enter "shut" to unlock it)
    Issuer name: CN=CA
    CA cert fingerprint: 1CB1376E 48B40FFC 8C68A493 D744A83A
    Granting mode is: auto
```

The following illustrates PKI trustpoint configuration, the IP address of the CA is 192.168.1.3.

Click here to view code image

```
Router(config)#crypto pki trustpoint PKI
Router(ca-trustpoint)#enrollment url http://192.168.1.3
Router(ca-trustpoint)#exit
Router(config)#crypto pki authenticate PKI
Certificate has the following attributes:
       Fingerprint MD5: 1CB1376E 48B40FFC 8C68A493 D744A83A
      Fingerprint SHA1: 805741B2 A094DFC3 CAFF1D7D 819EA99F B9C7879E

% Do you accept this certificate? [yes/no]: yes
Trustpoint CA certificate accepted.
Router(config)#crypto pki enroll PKI
%
% Start certificate enrollment ..
% Create a challenge password. You will need to verbally provide this
```

```
   password to the CA Administrator in order to revoke your certificate.
   For security reasons your password will not be saved in the configuration.
   Please make a note of it.

Password:
Re-enter password:

% The subject name in the certificate will include: CA.cisco.com
% Include the router serial number in the subject name? [yes/no]:
no
% Include an IP address in the subject name? [no]:
Request certificate from CA? [yes/no]: yes
% Certificate request sent to Certificate Authority
% The 'show crypto pki certificate verbose PKI' command will show the fingerprint.

Router(config)#
CRYPTO_PKI:  Certificate Request Fingerprint MD5: 552781FF 2584FBB1 9E014F06
51BCB61D
CRYPTO_PKI:  Certificate Request Fingerprint SHA1: 70190AD2 B255D1A9 03E942F8
A5F97C1F 5078E579
Router(config)#
%PKI-6-CERTRET: Certificate received from Certificate Authority
```

IPsec Configuration

This section describes the Cisco IOS IPsec configuration required to setup an IPsec session to or from a Cisco IOS router.

Cisco IOS IPsec implementation is interface-centric, which means that IPsec configuration, Security Policy Database (SPD), and Security Association Database (SAD) are per-interface. The term "IPsec interface" used in this book implies an interface with IPsec enabled on it by either the legacy **crypto map** or the newer **tunnel protection** method. Tunnel protection is a new way of configuring IPsec on Cisco IOS tunnel interfaces and is used with most of the Cisco IOS IPsec VPNs, such as FlexVPN and DMVPN. Therefore, this section will explain configuring IPsec using tunnel protection.

Any IPsec configuration should specify three key pieces of information:

- What traffic to protect (that is, the traffic selectors).
- How to protect that traffic (that is, the IPsec parameters to use, such as the IPsec protocol, mode, algorithms, SA lifetime, and others).
- Whom to protect the traffic with (that is, the peer).

With tunnel protection, the IPsec parameters are derived from the IPsec profile attached to the tunnel interface using the **tunnel protection** command, and the traffic selector and peer information is derived from the tunnel interface configuration as described below:

- The IKE/IPsec Security Association local address is derived from the **tunnel source** *ip-address* command on the tunnel interface.
- The IKE/IPsec Security Association peer address is derived from the **tunnel destination** *ip-address* command on the tunnel interface for point-to-point tunnel interfaces such as GRE (**tunnel mode gre**) or IPsec VTI (**tunnel mode IPsec**) or is learned dynamically for multipoint tunnel interfaces such as mGRE (**tunnel mode gre multipoint**).
- The SA traffic selectors for GRE interfaces are **ip gre host host** and for IPsec VTI **ip any any**.

IPsec Profile

The IPsec profile is the central configuration in IPsec that defines most of the IPsec parameters such as the protocol (Encapsulation Security Payload, Authentication Header), mode (tunnel, transport), algorithms (encryption, integrity, Diffie-Hellman), perfect forward secrecy (PFS), SA lifetime, and key management protocol (IKEv1, IKEv2).

Table 5-15 lists IPsec profile key commands and their usage.

IPsec Profile Commands	Usage
set transform-set	Specify an IPsec transform set that defines protocol, mode, and algorithms.
set security-association	Specify IPsec Security Association parameters such as lifetime, anti-replay settings, or idle-time.
set mixed-mode	Enable an IPsec Security Association between IPv4 peers to carry IPv6 traffic and vice versa
set reverse-route	Enable adding a static route to IP subnet protected by a remote IPsec peer, based on the negotiated SA traffic selectors.
set ikev2-profile	Specify IKEv2 as the key management protocol to negotiate the IPsec Security Associations and specify the IKEv2-profile to use for the negotiation.
set pfs	Enable perfect forward secrecy for the IPsec Security Association, where a Diffie-Hellman exchange is done during IPsec Security Association rekey to calculate the IPsec session key,
responder-only	Disable IPsec Security Association initiation from this device.

Table 5-15 *IPsec Profile Commands*

IPsec Configuration Example

The following example shows an IPsec transform-set, IPsec profile, and tunnel protection configuration. The IPsec transform set and IKEv2 profile are referenced from IPsec profile, which is then attached to tunnel interface with the **tunnel protection** command.

Click here to view code image

```
crypto ikev2 profile ikev2_profile
 match identity remote any
 authentication local pre-share key key1
 authentication remote pre-share key key1

crypto ipsec transform-set transform_set esp-aes esp-sha-hmac
 mode transport

crypto ipsec profile ipsec_profile
 set transform-set transform_set
```

```
 set ikev2-profile default

interface Tunnel0
 ip address 192.168.0.1 255.255.255.0
 tunnel source 172.16.1.1
 tunnel destination 172.16.2.1
 tunnel protection ipsec profile ipsec_profile
```

Smart Defaults

Smart defaults minimize IKEv2 and IPsec configuration by using default values wherever possible that cater to most of the use cases and can be customized to meet the needs of specific users. Smart defaults consist of the default IKEv2 proposal, default IKEv2 policy, and default IPsec profile. With smart defaults, a user just needs to configure the IKE2 profile specifying the authentication methods and the IKEv2 keyring or PKI trustpoint specifying authentication credentials and then apply the default IPsec profile to a tunnel interface. The following example shows the minimal IKEv2 and IPsec configuration required to create a basic IKEv2 and IPsec session when smart defaults are used.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key key1
 authentication remote pre-share key key1

interface Tunnel0
 ip address 192.168.0.1 255.255.255.0
 tunnel source 172.16.1.1
 tunnel destination 172.16.2.1
 tunnel protection ipsec profile ipsec_profile
```

The default IKEv2 proposal, IKEv2 policy, and IPsec profile can be displayed by issuing the following **show** commands. Note that the default IPsec profile references an IKEv2 profile with name *default*, as highlighted in the **show** command output below; hence an IKEv2 profile with the name "*default*" must be defined.

Click here to view code image

```
Router#show crypto ikev2 proposal default
 IKEv2 proposal: default
     Encryption : AES-CBC-256 AES-CBC-192 AES-CBC-128
     Integrity  : SHA512 SHA384 SHA256 SHA96 MD596
     PRF        : SHA512 SHA384 SHA256 SHA1 MD5
     DH Group   : DH_GROUP_1536_MODP/Group 5 DH_GROUP_1024_MODP/Group 2

Router#show crypto ikev2 policy default
 IKEv2 policy : default
      Match fvrf : any
      Match address local : any
      Proposal    : default

Router#show crypto ipsec profile default
IPSEC profile default
        IKEv2 Profile: default
        Security association lifetime: 4608000 kilobytes/3600 seconds
        Responder-Only (Y/N): N
        PFS (Y/N): N
        Mixed-mode : Disabled
```

```
Transform sets={
        default:  { esp-aes esp-sha-hmac  } ,
}
```

Summary

This chapter introduces the foundational IKEv2 configuration that is needed for any IKE-based VPN, whether it is point-to-point, remote-access or hub-spoke. This chapter provides essential background information for understanding more advanced topics such as IKEv2 advanced features, FlexVPN Client, FlexVPN Server, and IKEv2 load balancing. The key point to remember from this chapter is the concept that the IKEv2 profile is an anchoring point for most of the IKEv2 configurations and features.

Chapter 6. Advanced IKEv2 Features

IKEv2 is a modular protocol and has the ability to incorporate a number of additional tweaks and features into the protocol itself.

The industry has developed additional features into IKEv2, such as IKEv2 fragmentation that was added after the initial release of the RFC. Prior to the standardization of IKEv2 fragmentation, it was implemented as a Cisco proprietary feature in Cisco IOS.

Cisco has developed a number of other proprietary features that interact with IKEv2 and enhance the architecture of VPNs. IKEv2 allows for the transportation of security group tags (SGT) when using Cisco TrustSec.

Cisco IOS allows for the handling of expired or revoked certificates in a strict manner, allowing for certificates that have been used to authenticate an IKEv2 SA, but are then revoked or expired, to remove the IKEv2 SA.

Introduction to IKEv2 Fragmentation

IKEv2 uses UDP as the transport protocol for its messages. The majority of IKEv2 messages are generally small in size, with most being well below 500 bytes. The exception to this is the IKE_AUTH exchange. When using certificates, the size of the messages used in the exchange can become large, in some cases too large for traffic to transverse certain links. When the size of the IP packet (aka datagram) is too large, a number of scenarios can occur with regard to the router that owns the link:

- The IP packet can be dropped and a message indicating this condition is sent back to the sender.
- The IP packet is *split* (fragmented) into smaller pieces so that these can fit on the link.
- The IP packet can be discarded and no action taken.

In many cases when IP packets are fragmented, problems can occur, such as fragmented packets being dropped by security or network address translation (NAT) devices, or the indication message that a packet was dropped is discarded en route to the sender.

Any of these scenarios can lead to loss of connectivity with the result that the IKEv2 SA is not established. The use of IKEv2 fragmentation mitigates these issues by performing fragmentation at the IKE layer. If using IKEv2 fragmentation, an IKE message will be *split* into IKE fragments, and each of these is then sent alone. The sending of these single IKE fragments ensures that when the IKE traffic is encapsulated within UDP and then IP, the size of the resulting IP packet will never exceed the smallest link size between IKE peers.

IP Fragmentation Overview

The IP protocol can operate over a wide variety of transmission links and media types. Theoretically the maximum length of an IP packet is 65535 bytes; this is the largest IP packet size possible. However, due to the fact that most transmission links enforce a smaller maximum packet length limit, also known as the maximum transmission unit (MTU), this value is rarely achieved. The exact value of the IP MTU depends on the type of the transmission media or interface type. It is common for networks of various media types to have different sized MTUs, and even when networks have the same transmission type (such as Ethernet), IP MTUs can differ between equipment types and versions.

Note

Fragmentation can only occur if the Don't Fragment (DF) flag is not sent within the IP header.

Should a router receive an IP packet that is too big to forward out of an interface because the local interface has a lower IP MTU than the size of the IP packet, and if the DF flag is not set, then the IP packet must be fragmented. Fragmentation involves breaking the packet into a number of pieces so that the new size of the packets is within the size permitted by the interface that the packet will exit. When the original packet is fragmented, each newly created packet will have a new total length, fragment offset, and more fragments field.

When the intended recipient receives the fragmented packets, it can reassemble them, using the fragment offset and more fragments field contained in the IP header. This is detailed in Chapter 15 "IPsec Overhead and Fragmentation." The issue with network devices performing fragmentation is that this is very resource intensive; on almost all devices, fragmentation is performed in software that requires CPU and memory resource. The receiving device also needs to allocate memory and resources to reassemble the received fragments.

Hosts do have a method to detect the MTU in use on a path and can adjust their MTU for a session so that fragmentation will not occur. The method, known as path MTU discovery (PMTUD), allows for a host to set the DF flag within the fragmentation field of the IP header. Setting the DF flag will ensure that if an IP datagram is required to be fragmented by a network device en-route, fragmentation will not occur. Instead the IP datagram will be discarded, and the network device discarding the datagram will send an Internet control message protocol unreachable message, indicating that the fragmentation was needed, however the DF bit was set (ICMP Type 3 Message 4). This message will contain the IP header of the datagram that required fragmentation along with the first 64 bytes of the payload and the largest MTU that can be sent on the link. When the host receives this ICMP message, it can then reduce the MTU for this session to ensure that future IP datagrams are not fragmented. This process can occur for any links between hosts that requires traffic to be fragmented.

Some network devices can be configured to drop traffic that requires fragmentation and not send an ICMP unreachable message. In this instance, the concept of PMTUD becomes broken, resulting in failed connectivity between hosts. Many network security devices are configured to drop ICMP messages. A network device might drop an IP datagram, which requires fragmentation but has the DF bit set, and send a ICMP unreachable message. The ICMP message might be dropped en-route to the recipient, with the result that all traffic requiring fragmentation is not received by the intended destination.

IKEv2 and Fragmentation

The IKE_SA_INIT message can become large in size should a Diffie-Hellman group be used with a large public key (such as Group 18 which uses an 8192-bit MODP). This can considerably add to the size of the resulting IP datagram. The IKE_AUTH exchange can contain one or more certificates, which can result in the packet size exceeding over 1000 bytes. Depending on the public key used for the certificate and the number of certificates sent (if chaining is used), the resulting packet can commonly exceed 1500 bytes, which is the default largest IP MTU on most interfaces. Another common cause of fragmentation in IKEv2 is heavily populated configuration payloads containing many attributes. Cisco AnyConnect advertises the fact that it supports many features using configuration payload, which commonly results in a single IKE_SA_INIT or IKE_AUTH message being fragmented into six or more fragments.

Figure 6-1 illustrates an IKE message being fragmented at the IP layer. As the IKE message is larger than the IP MTU, this message is fragmented at the IP layer, resulting in a number of IP datagrams being created to send the single IKE message.

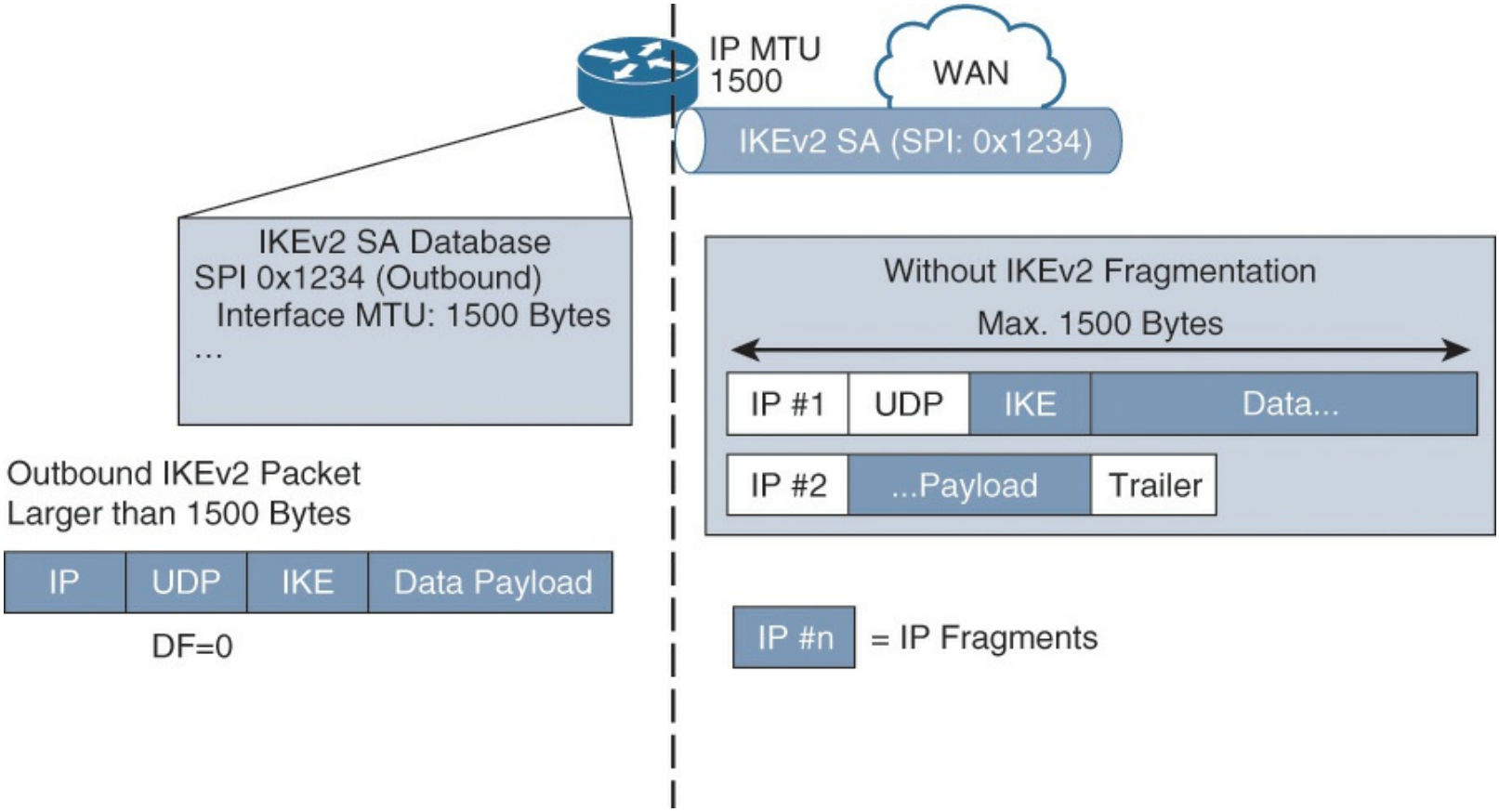


Figure 6-1 *IP Fragmentation of an IKE Message*

In November 2014, RFC7383—Internet Key Exchange Protocol Version 2 (IKEv2) Message Fragmentation—was released, which describes how to negotiate and fragment IKEv2 packets. RFC7383 describes how each IKEv2 peer will advertise the ability to fragment IKEv2 traffic by sending an IKEV2_FRAGMENTATION_SUPPORTED notification payload (type 16430) within the IKE_SA_INIT exchange.

Assuming that both peers include these payloads, the IKE_AUTH and subsequent encrypted exchanges can be fragmented at the IKE layer, rather than the IP layer. IKEv2 messages can be configured to be fragmented should the size of the IKEv2 payload exceed a defined value. Fragmented IKEv2 messages are sent individually within an Encrypted Fragment payload (defined in RFC7383);

the Encrypted Fragment payload contains additional fields to indicate that the packet is a fragment and should be reassembled once successfully decrypted and authenticated by the receiving device.

Figure 6-2 illustrates the negotiation of the ability to support fragmentation. This will only occur within the IKE_SA_INIT exchange, and the fragmentation will only then be performed on IKEv2 traffic that is to be protected.

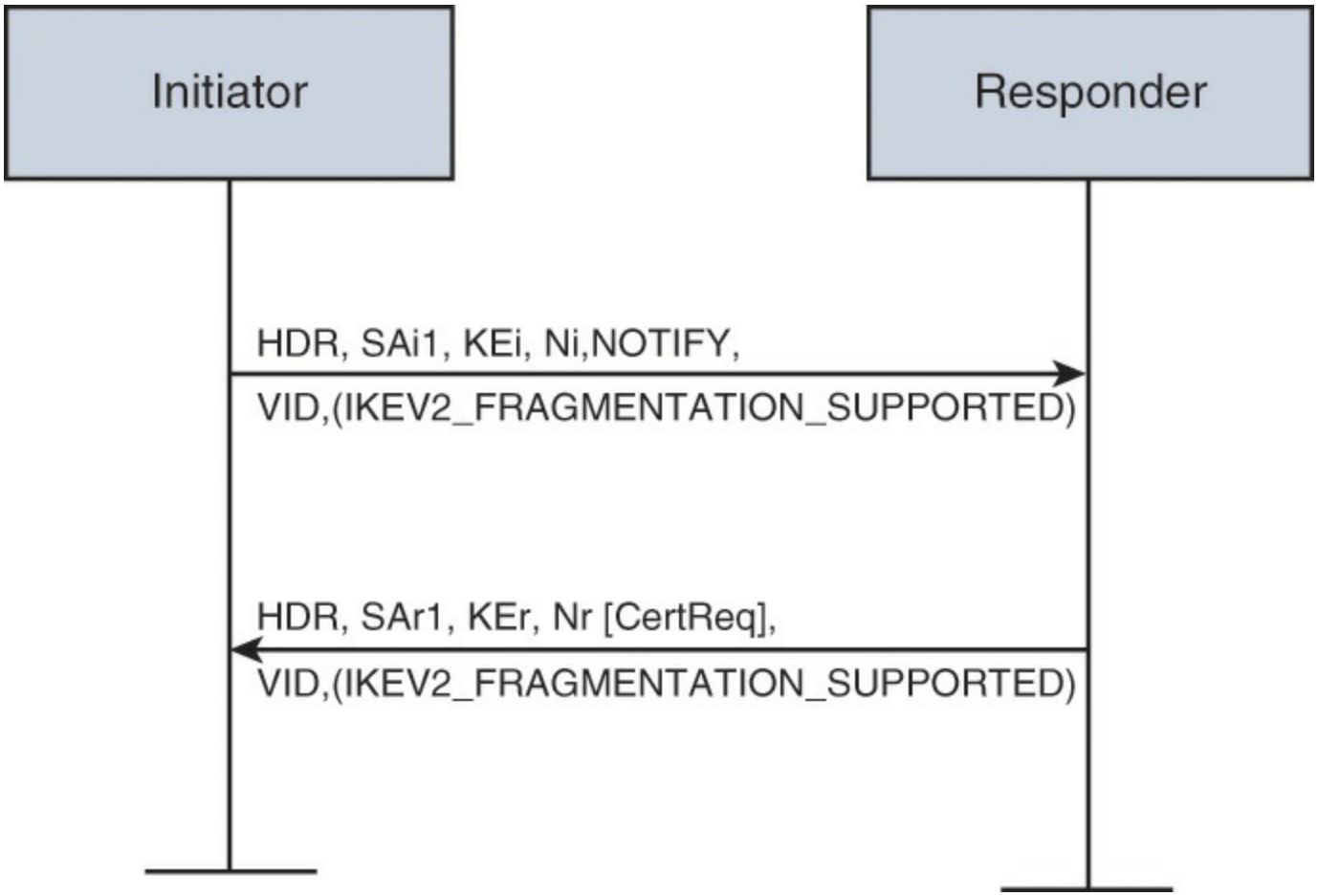


Figure 6-2 *IKE_SA_INIT Exchange with Fragmentation Capability*

Note

Allowing IKEv2 fragmentation mitigates the chance of allowing Denial-of-Service (DoS) attacks which utilize weaknesses in fragment reassembly in hosts.

Figure 6-3 illustrates the Encrypted Fragment payload used when IKEv2 fragmentation is activated. The fragment number and total fragments fields in the Encrypted Fragment payload are used by the receiver to reconstruct the IKEv2 message if fragmentation occurs.

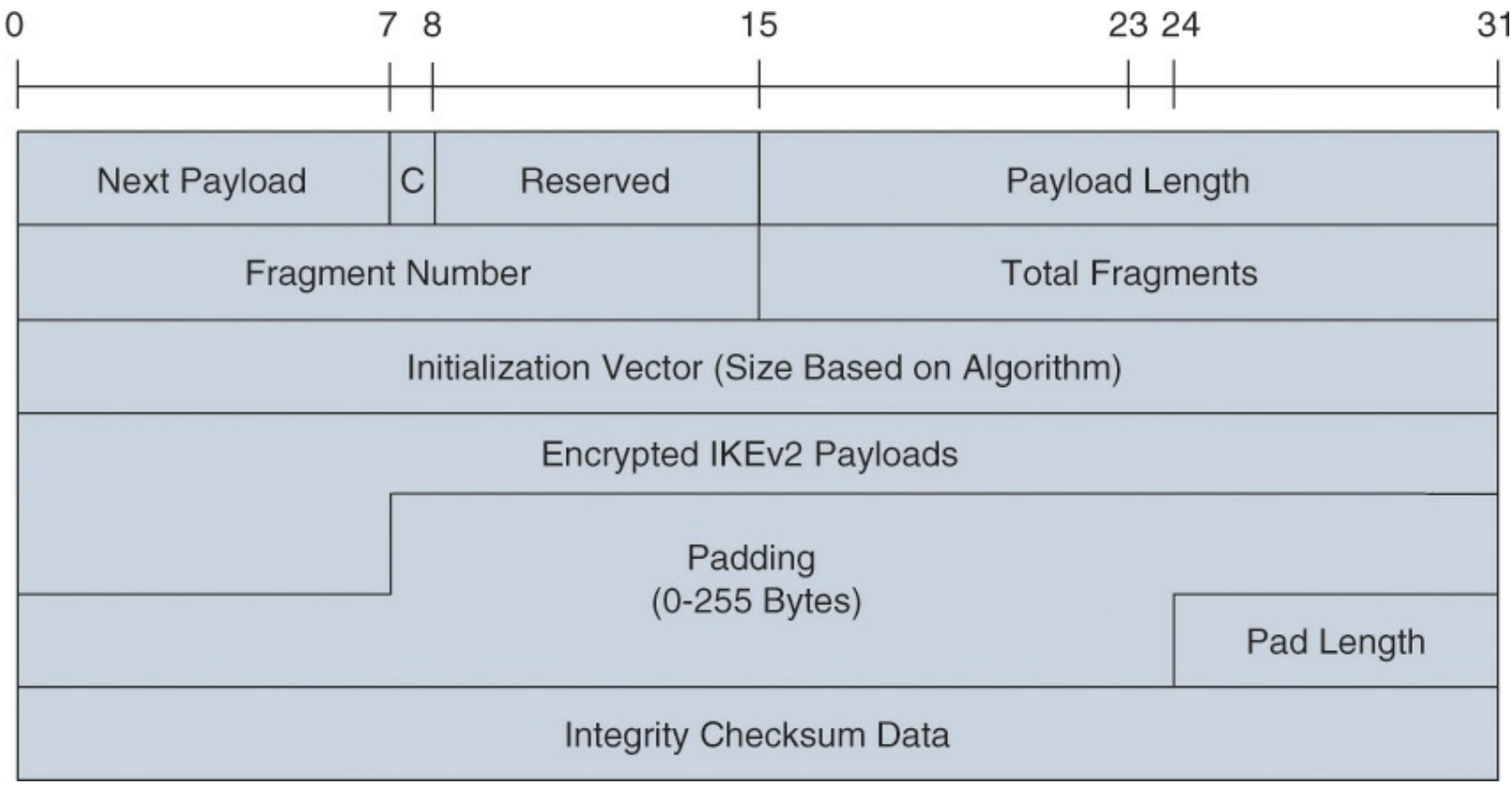


Figure 6-3 *Encrypted Fragment payload*

For example, if an IKE_AUTH message is required to be 1400 bytes, with UDP (6 bytes) and IPv4 (20 bytes) overhead, this would result in an IP packet being 1426 bytes. Should this packet encounter a link with an IP MTU of 1400 bytes and the DF bit was not set, the packet would be fragmented into two new IP packets.

If IKEv2 fragmentation were used where a value of 800 bytes was configured, then the total size of the IKEv2 message and IP and UDP headers would be 800 bytes. The actual size of the IKEv2 message would be 800 – 6 – 20 = 774 bytes. Rather than sending a single IP packet 1426 bytes in size, two IP packets of size of 800 bytes or less would be sent without the need for fragmentation at the IP layer. Within the Encrypted Fragment payload, the first message would have a fragment number of 1, with the total fragments field set to 2; the second message would have a fragment number of 2 with the total fragments field set to 2. The receiver would decrypt and verify all three received IKEv2 packets and then reconstruct the 1400 byte IKE_AUTH message to then process.

> **Note**
>
> As the size of the IKEv2 packet is configured, the minimum IP MTU between the peers must be known prior to fragmentation at the IKE layer (to calculate the minimum IKEv2 message size). If it is not known, then a value must be guessed that will be below what will be fragmented at the IP layer.

Figure 6-4 illustrates how an outgoing IKE message is fragmented at the IKE layer on an IOS device when IKEv2 fragmentation is enabled with a MTU of 1000 bytes. An IKEv2 message with a size over 1000 bytes is sent out as multiple separate IP payloads, which are not fragmented. The receiver will re-assemble the IKE payload on receipt.

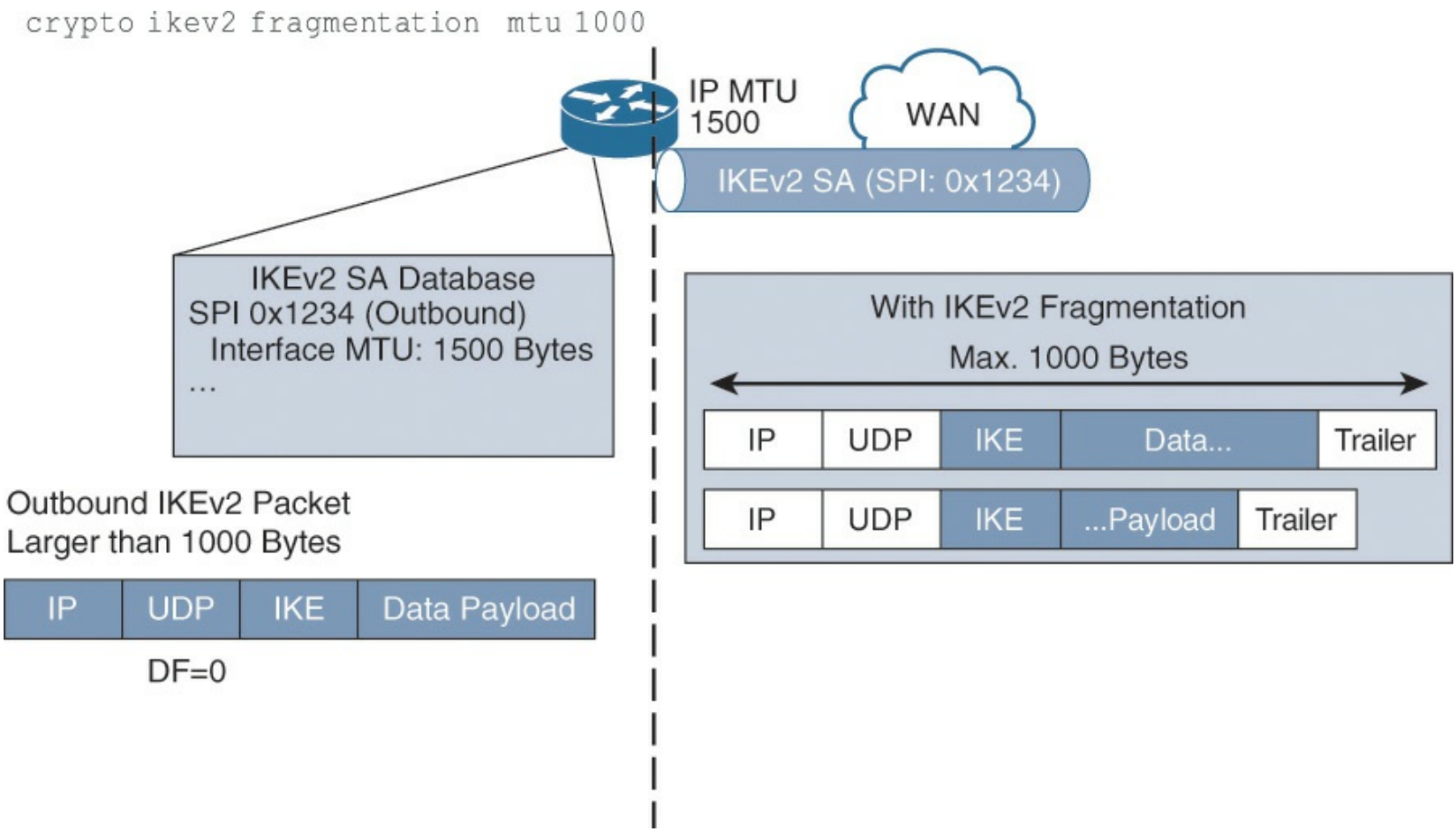


Figure 6-4 *IKEv2 Fragmentation*

For many years, Cisco IOS has supported the use of fragmenting IKEv2 traffic, using a Cisco proprietary feature, similar to the IETF version but slightly different in the method used to negotiate the capability. Cisco proprietary fragmentation is performed after encryption, while IETF fragmentation is before encryption. The IETF version uses a new Encrypted Fragment payload which has the Fragment Number and Total Fragments fields in cleartext, so decryption is not required to validate the Fragment Number and Total Fragments fields on the receiver.

With Cisco fragmentation the Fragment Number and Total Fragments fields are sent encrypted, so decryption is required before these fields can be validated. The Cisco version uses the Vendor ID payload to advertise the capability; this is contrary to the IETF version that uses the notification payload to announce the ability to use IKEv2 fragmentation.

Cisco IOS 3.15, 15.5.2T, and onward have the ability to use either the standardized IETF version or the Cisco proprietary version. If configured to perform IKEv2 fragmentation, Cisco devices will advertise both IETF and proprietary capabilities. By advertising both methods, Cisco IOS post 3.15/15.5.2T can interoperate with peers that support only the Cisco proprietary method, the IETF standard method, or both.

Table 6-1 details the behavior of IKEv2 fragmentation on an Cisco IOS device post 3.15. If both the Cisco and IETF methods are advertised, then the IETF method will be selected. This allows for Cisco devices to interoperate with older Cisco devices that only support the proprietary method, as well as Cisco or third-party devices that support the standardized method.

Peer1 Capability	Peer2 Capability	Active Frag Type on SA
IETF and CISCO	IETF and CISCO	IETF
IETF	IETF	IETF
CISCO	CISCO	CISCO
CISCO	IETF and CISCO	CISCO
IETF	IETF and CISCO	IETF
IETF	CISCO	NONE
NONE	IETF/CISCO	NONE

Table 6-1 *IKEv2 Fragmentation Capabilities*

By default, IKEv2 fragmentation is not enabled on Cisco IOS; to enable IKEv2 fragmentation, the global command **crypto ikev2 fragmentation** must be enabled.

The default IKEv2 fragmentation MTU size is set to 576 bytes for IPv4 transport and 1280 bytes for IPv6 transport. This value will set the size of the IKEv2 message plus the UDP and IP overhead, which is calculated at 8 bytes for UDP, 20 bytes for IPv4, and 40 bytes for IPv6.

To configure IKEv2 fragmentation, the command **crypto ikev2 fragmentation mtu *mtu-size*** is used, where the MTU size is from 68 to 1500 bytes.

To verify if an IKEv2 session is using fragmentation, issue the **show crypto ikev2 sa detailed** command. This will indicate if fragmentation is used and the method. The following output illustrates an example output where the use of IKEv2 fragmentation can be seen, along with the type of fragmentation used, in this case it being the standard IETF method.

Click here to view code image

```
Router#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         10.10.10.1/500        10.10.50.5/500        OUTSIDE/none         READY
      Encr: AES-CBC, keysize: 128, PRF: SHA256, Hash: SHA256, DH Grp:14, Auth
sign: RSA, Auth verify: RSA
      Life/Active Time: 120/38 sec
      CE id: 1074, Session-id: 1171
      Status Description: Negotiation done
      Local spi: F78AC1F7C7C9F558       Remote spi: 5043A7F7F7E1A622
      Local id: hostname=R1.cisco.com
      Remote id: hostname=R5.cisco.com,cn=r5,o=cisco,ou=service1
      Local req msg id:  0              Remote req msg id:  3
      Local next msg id: 0              Remote next msg id: 3
      Local req queued:  0              Remote req queued:  3
      Local window:      5              Remote window:      5
      DPD configured for 0 seconds, retry 0
      IETF Std Fragmentation  enabled.
      IETF Std Fragmentation MTU in use: 972 bytes....
```

IKEv2 SGT Capability Negotiation

Security group tagging (SGT) is a technology that overcomes the shortcomings of the traditional approaches to network control policy administration. When a user or device connects to the network and sends network traffic, a Cisco access switch has the ability to automatically profile the user or device and then tag all incoming traffic, based on a policy applied to the profile. 802.1x in conjunction with RADIUS is commonly used to profile the user and/or device. The tag or Security group tag (SGT) is a numerical value and is either manually assigned to the access switches or automatically administered through the use of the Cisco identity services engine (ISE) and applied to network traffic meeting the profile. This tag can then be used on any capable networking device to make access control policy decisions.

Figure 6-5 illustrates the use of a SGT within a basic network. Traffic is classified from each host by the network access switch;, this traffic is then applied a tag when egressing the switch. An access-control-list (ACL) is applied on the router that permits various tagged traffic. This ACL will filter traffic based on SGT rather than the standard 5-tuples (source/destination IP address, protocol and ports) of information.

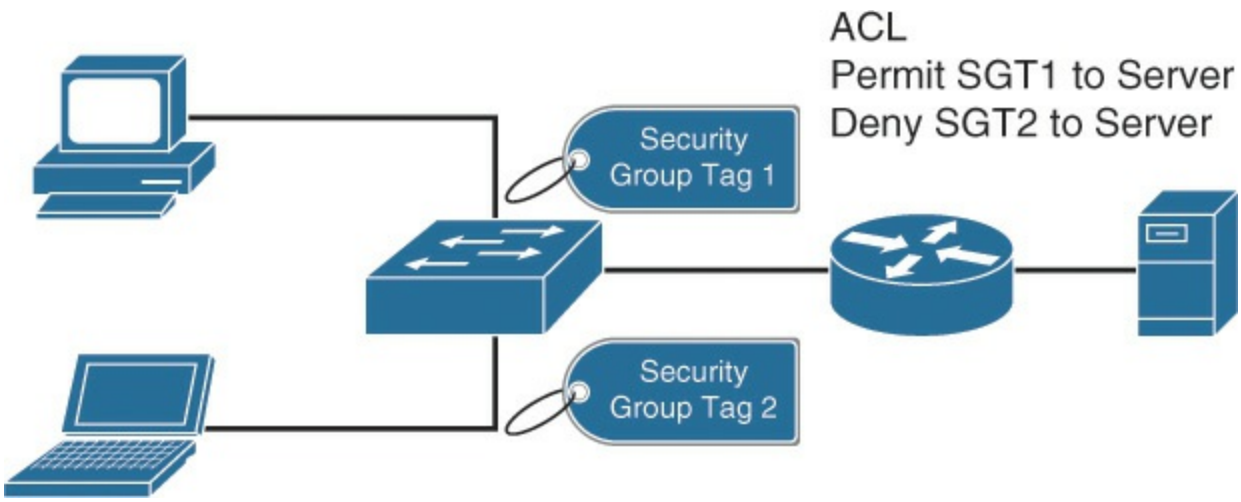


Figure 6-5 *Security Group Tagging*

In order to pass the information about the SGT tag, the Ethernet frame is amended with an additional field, known as Cisco meta data (CMD). Figure 6-6 details the structure of the CMD header. Although the SGT is included within the CMD, many people still refer to the CMD as the SGT.

This CMD field is added to the Ethernet header, between the source MAC and the Ether Type fields. This modification can be thought of as similar to the way modifications are made for 802.1q tags. The modified Ethernet frame can be understood only by selected Cisco devices and will not be passed on devices that do not support or understand this technology.



Figure 6-6 *Cisco Meta Data header*

Figure 6-6 illustrates a Cisco Meta Data header.

When network devices that support SGTs are interconnected by network devices that don't have this ability (and are unable to read the CMD header), a protocol exists to transfer information about SGT bindings. The Security group tag exchange protocol (SXP) is used by network devices to propagate

information about IP to SGT bindings. SXP allows for non-adjacent networking devices to inform each other about IP to SGT mappings that were created statically or assigned dynamically to clients after profiling.

Cisco IOS allows for an IPsec Security Association to include the CMD header within the Encapsulation Security Payload header. Note that this is a Cisco proprietary feature and, at the time of writing, is only available on Cisco IOS. The CMD header is included after the Encapsulation Security Payload header, resulting in the SGT being encrypted and integrity protected. There are a number of benefits for including the SGT tag within an IPsec Security Association:

- The SGT allows for policy-based decisions to be made on the protected traffic by the IPsec peer.
- The SGT can be viewed by the receiver, which allows for firewall or IPS functionality to then be applied on incoming traffic should these features be enabled.
- Having the SGT included within the IPsec Security Association also allows for the SGT to be passed between devices capable of understanding the SGT, without the need for running SXP.

The ability for a device to support including the SGT within an IPsec Security Association is negotiated within the IKE_SA_INIT exchange. Devices advertise the fact that they support this ability using the Vendor ID (VID) payload of type CISCO-CTS-SGT. This is a Cisco private VID and should be ignored by any devices that don't support this feature.

If the negotiation did not occur and the SGT was included by one IPsec peer and the opposing peer did not support or understand the CMD, the decrypted traffic would be dropped. The Encapsulation Security Payload header indicates the next header is CMD by the use of the next header being protocol 99 (any private encryption scheme).

Once both peers have advertised the fact that they support the sending of SGTs within the IPsec Security Association, when clear text traffic enters the tunnel interface—if there is an IP to SGT mapping—the relevant SGT will be added to outgoing traffic prior to encryption. If there is no IP to SGT mapping then a SGT of 0 will be used in the CMD.

Figure 6-7 illustrates the inclusion of the CMD within Encapsulation Security Payload.

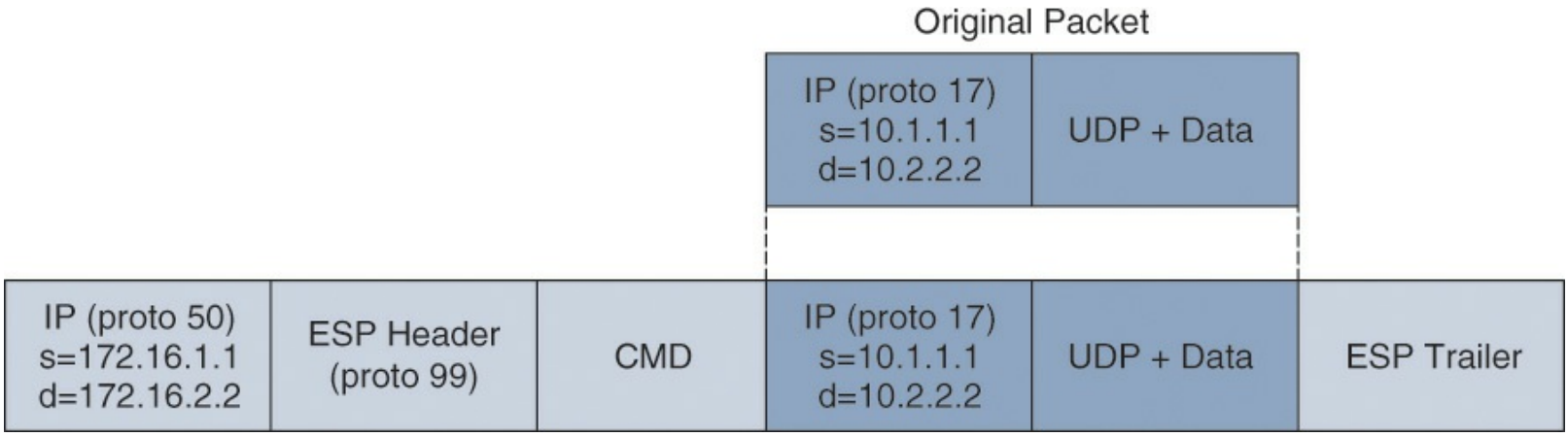


Figure 6-7 *IPsec Tunnel Mode with CMD header*

It should be noted that, at the time of writing, the use of SGT within an IPsec Security Association is only available for IPsec using VTIs, the reason being when using tunnel mode GRE, traffic will be encapsulated within GRE prior to being encrypted. The tunnel will only see the GRE source and destination address and so will not be able to match on the IP to SGT mapping (which will be based on end-host IP addresses) because it is currently unable to view the traffic that is traversing the

tunnel. Using tunnel mode GRE will not enable the ability to see the internal traffic; this is currently tracked via Cisco bug ID CSCuj25890.

> **Note**
>
> If a packet is fragmented either before or after encryption, only the first fragment will be inline tagged with the SGT.

The following output illustrates how to verify that an IPsec Security Association has included a SGT.

Click here to view code image

```
Router#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         10.10.10.1/500        10.10.50.5/500        OUTSIDE/none         READY
      Encr: AES-CBC, keysize: 128, PRF: SHA256, Hash: SHA256, DH Grp:14, Auth
sign: RSA, Auth verify: RSA
      Life/Active Time: 120/38 sec
      CE id: 1074, Session-id: 1171
      Status Description: Negotiation done
      Local spi: F78AC1F7C7C9F558       Remote spi: 5043A7F7F7E1A622
      Local id: hostname=R1.cisco.com
      Remote id: hostname=R5.cisco.com,cn=r5,o=cisco,ou=service1
      Local req msg id:  0              Remote req msg id:  3
      Local next msg id: 0              Remote next msg id: 3
      Local req queued:  0              Remote req queued:  3
      Local window:      5              Remote window:      5
      DPD configured for 0 seconds, retry 0
Fragmentation not configured.
      Extended Authentication not configured.
      NAT-T is not detected
      Cisco Trust Security SGT is enabled
```

Verbose debugging can be enabled to verify the SGT applied to traffic using the **debug crypto IPsec metadata sgt** command.

IKEv2 Session Authentication

As the IKE_AUTH exchange only occurs once when creating an IKEv2 session between peers, and within this exchange, both peers authenticate each other should the authentication method used be digital-signatures, one or more certificates will be exchanged. At the time that this authentication occurs, a number of checks will be performed on the presented certificate in addition to the verification of the digital signature:

- Cryptographic checks are performed to ensure that the certificate was signed by the private-key of the issuing certificate authority.
- The certificate validity period will be checked to ensure that the certificate has not expired.
- If a revocation check is enabled, the serial number of the certificate will be checked to confirm that this has not been revoked. This can be via online certificate status protocol (OCSP) or checking a certification revocation list (CRL).

Once these checks have occurred, at any point in the future should the certificate presented by the peer

become revoked or expires, there is no inbuilt mechanism within IKEv2 to then re-authenticate the session to perform the certificate revalidation.

Note that the experimental RFC, RFC4478 “Repeated Authentication in Internet Key Exchange (IKEv2) Protocol,” describes the ability to perform repeated authentication within IKEv2. However, this is not implemented by Cisco IOS.

The behavior of IKEv1 is different from IKEv2 in the regard of re-authentication. Within IKEv1 an authentication exchange will occur on the IKEv1 session rekeying.

IKEv2 Session Deletion on Certificate Revocation

When a certificate authority issues a certificate, this certificate is used by the owner of the private key that relates to the public key contained within the certificate. The purpose of a certificate is to cryptographically bind an identity to the public key. Should the private key become lost or stolen, then there is no guarantee that whoever presents the certificate and is in possession of the private key is who the certificate classes as the identity.

A certificate authority supports the ability to revoke certificates that for some reason become untrusted. This could be if the private key is lost/stolen or if the certificate was issued incorrectly. A certificate revocation list (CRL) is a list of serial numbers of certificates that have been revoked by the certificate authority. This list is cryptographically signed by the certificate authority and includes a validity period.

Cisco IOS allows for the ability such that if CRL checking is enabled, when a CRL is downloaded, a check will be performed on the serial numbers of all certificates that were used in the authentication of all active IKEv2 SAs. Should the serial number of a certificate be included in the CRL for an active IKEv2 session, that session will be instantly deleted.

To activate the ability to disconnect a revoked peer, the **crypto ikev2 disconnect-revoked-peers** should be used.

Figure 6-8 illustrates the behavior of IKEv2 session deletion on certificate revocation, when a client connects to an IPsec gateway. When CRL checking is enabled, the CRL will be checked on the initial IKEv2 connection. At this point, the CRL could be locally cached or will need to be downloaded. The serial number of the certificate presented by the client is checked against the CRL by the gateway; at this point the serial number of the certificate is not within the CRL so the authentication is successful. If at some point later, the client's certificate is revoked, when the updated CRL is then downloaded by the gateway, the gateway will check the serial numbers of all existing IKEv2 SAs, including the sessions that were established earlier, using the client certificate which is then deleted.

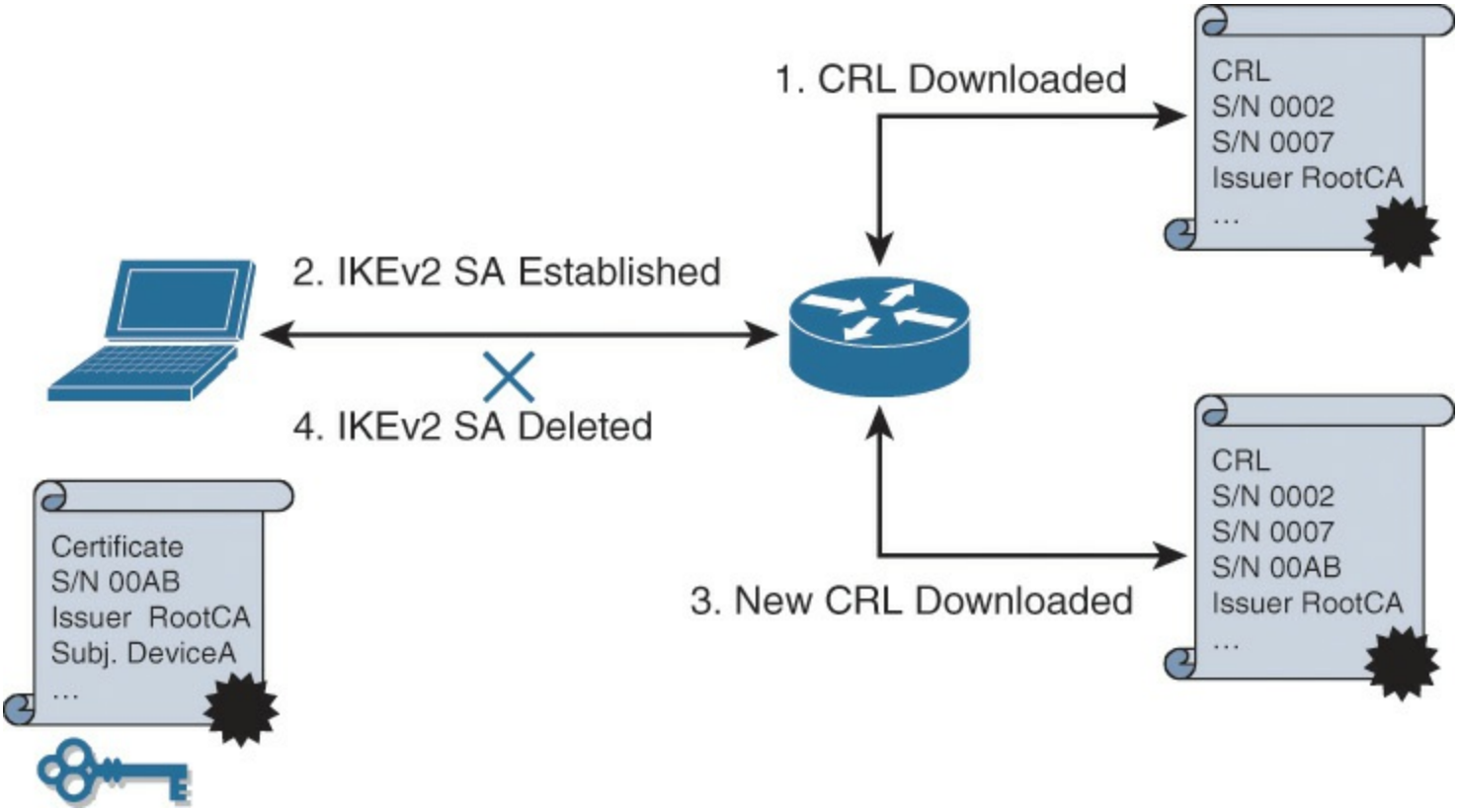


Figure 6-8 *Session Deletion on Certificate Revocation*

An INFORMATIONAL message will be generated to inform the peer that the session is deleted, containing the error code IKEV2_DELETE_CERT_REVOKED.

The following syslog message is generated when an existing IKEv2 session is deleted due to a certificate being revoked;

Click here to view code image

```
IKEv2:Sending Delete Reason Notify with error code: IKEV2_DELETE_CERT_REVOKED
severity: INFORMATIONAL
```

The ability to delete a session that is using a revoked certificate after the session has been established allows for Cisco IOS implementations to adhere to many companies' security policies which dictate that a certificate must be revoked with a certain period of time and that any VPNs created with that certificate must also be removed within a certain period.

A CRL will have a valid lifetime. If, for example, the lifetime of a CRL is set to 24 hours and an IPsec gateway downloads the CRL one hour after it was published, the IPsec gateway would cache the CRL for the remaining lifetime (23 hours). If a certificate is revoked two hours after the CRL has been downloaded, then the remaining lifetime (21 hours) of the CRL could be used by an attacker to exploit the compromise of a private key by connecting to the VPN gateway before the new CRL containing the revoked certificate was downloaded.

A number of options can be used to reduce the time that a CRL will remain cached on Cisco IOS and before a new CRL obtained:

- Download the CRL at regular intervals, using the command **crypto pki crl download schedule time**.
- Use the command **crypto pki crl download schedule prepublish** to download the CRL prior to the expiry lifetime. This can be used to ensure that the CRL is downloaded at a time prior to the

expiry time.

- Refresh the CRL cache at regular intervals by renewing all CRLs that are cached (this requires that the cache is populated). This could be scripted using the commands **crypto pki crl download trustpoint** and **crypto pki crl refresh-cache**.

IKEv2 Session Deletion on Certificate Expiry

When a certificate authority issues a certificate, this certificate will have a defined lifetime, which is the time that the certificate can be used for. This lifetime is denoted by the certificate validity period, which is the time interval during which the CA warrants that it will maintain information about the status of the certificate. An example of maintenance is that the certificate will only be revoked within this lifetime. The validity field is represented as two dates, the date on which the certificate validity period begins (notBefore) and the date on which the certificate validity period ends (notAfter).

As the authentication within IKEv2 only occurs once, if a certificate is used to authenticate a client and the current time is within the validity period, this certificate can successfully be used. If the current time is outside of the validity period, then authentication will fail.

This opens the interesting case for an IKEv2 SA that is established using a certificate within the validity period but is very close to expiring. Once the certificate expires (the validity period passes), then prior to Cisco IOS version 15.4(3)M or 3.13, there was no automatic method to remove an IKEv2 SA which was created with a certificate.

Since Cisco IOS version 15.4(3)M or 3.13, there is now the ability for the lifetime of the IKEv2 SA to be tied to the lifetime of the certificate validity period. This ensures that once a certificate is outside of the validity period, then the IKEv2 SA is torn down, and for an IKEv2 SA to be re-established, re-authentication will be required (which would then fail as the certificate is expired). This behavior is enabled by default within the IKEv2 profile. Because the behavior is default, the command to enable this behavior **lifetime certificate** is hidden by default but can be seen by using the **show run all** command. Should this behavior be required to be disabled, the **no lifetime** command can be used within the IKEv2 profile to disable this behavior.

In Cisco IOS version 15.4(3)M and 3.13, when an IKEv2 SA is created or a subsequent IKEv2 rekey occurs, the certificate validity time is checked to determine if the IKEv2 SA lifetime is greater than the remaining validity time of the certificate. If the IKEv2 SA lifetime is greater than the remaining certificate validity time, the following IKEv2 debug message is generated, and the IKEv2 SA lifetime is adjusted to only last as long as the certificate validity time.

Click here to view code image

```
IKEv2:Adjusting lifetime due to Cert Validity date
```

The IKEv2 SA will have the adjusted lifetime, when the IKEv2 SA then meets this lifetime rather than an IKEv2 SA rekey occurring the IKEv2 SA is deleted, and the following debug message generated.

Click here to view code image

```
IKEv2:(SESSION ID = 24,SA ID = 1):Queuing IKE SA delete request reason:
certificate lifetime expired
```

IKEv2 Session Lifetime

In the process of establishing an IKEv2 session, both peers authenticate each other; this authentication occurs within the IKE_AUTH exchange. IKEv2 use the concept that once authentication has successfully completed, authentication does not occur again for the duration of the IKEv2 session. When new IKEv2 or IPsec Security Associations are required, these are generated using the CREATE_CHILD_SA exchange, which doesn't include any form of authentication.

This concept surprises many individuals who are familiar with IKEv1; in IKEv1, the session will have a lifetime (denoted in seconds). This lifetime is included within the policies used to negotiate the IKEv1 session (sent in MM1 and MM2). Just prior to the IKEv1 SA expiring, a new IKEv1 SA is created, which requires a fresh authentication exchange, and then the new IKEv1 SA is used. Having a fresh authentication is more computationally expensive; however, it brings the benefits of proving authenticity for every new IKE exchange.

An IKEv2 Security Association has a lifetime; however, this lifetime is held locally and not negotiated between parties. On Cisco IOS, the default lifetime for an IKEv2 SA is 86,400 seconds or 24 hours; this lifetime is configurable within the IKEv2 profile, using the lifetime command. Before the IKEv2 SA lifetime is reached, a peer will rekey the IKEv2 SA by initiating a CREATE_CHILD_SA exchange. This exchange will create a new IKEv2 SA with new SPIs and key material. This new IKEv2 SA will then exist for its lifetime and should rekey a new IKEv2 SA unless removed by a process or individual. Although each IKEv2 SA has a lifetime, because the IKEv2 SA is free to generate child IKEv2 SAs, the overall crypto session can potentially live forever.

The method to authenticate a peer can be computationally expensive; you could ask the question *if I have authenticated a peer once, why would I need to be validated again?* This is true in many cases; however, business needs might dictate that an IKEv2 session only lasts for a certain lifetime. If a guest or partner was given VPN access for a period of time, would the business expect that the session could last indefinitely? There might be a security requirement that authentication re-occur at least every 24 hours. In the modern age, it is common for mobile devices to be left on trains or go missing after a night at the pub. In this case, the behavior of IKEv2 to only authenticate once is not ideal as once an IKEv2 session is established, by default it will last until it is manually deleted, disconnected due to timeout, or loss of services results in premature session termination.

To allow for an IKEv2 session to have a strictly defined lifetime, Cisco IOS can be configured to set the lifetime of the cryptographic session when the IKEv2 SA is created. Once this lifetime has been met, the IKEv2 SA and all related IPsec Security Associations will be deleted. This strictly defined lifetime will ensure that the IKEv2 SA only occurs for the configured period and after that the IKEv2 SA is deleted (without any other IKEv2 SAs being rekeyed using the CREATE_CHILD_SA exchange). Should a new session be required between both peers, a new IKE_SA_INIT and IKE_AUTH exchange will occur, guaranteeing that the two peers will authenticate each other.

Having a strict session lifetime allows for scenarios, such as a user being granted access for a given period; when this period has elapsed, the session will be removed. Should the user attempt to re-authenticate, their access would have expired and authentication within the IKE_AUTH exchange would subsequently fail for the new session.

Without this behavior, should it be necessary to remove a session, the session would need to be identified and then cleared, using either of the following commands: **clear crypto ikev2 sa** or **clear crypto session**.

The session lifetime has been configurable since version 15.5(1)T or 3.14, where the lifetime can be obtained via the local IKEv2 authorization policy or a policy obtained from RADIUS. This lifetime is then used when creating the crypto session. If the session lifetime is less than the default IKEv2 SA lifetime, then the IKEv2 SA will only be created for the session lifetime, and once the IKEv2 SA expires, it is deleted. If the session lifetime is greater than the IKEv2 SA lifetime, the IKEv2 SA will be created, however subsequent IKEv2 SAs will only be rekeyed until the remaining session lifetime is less than the IKEv2 lifetime. At this point, the final IKEv2 SA will have a lifetime that ensures it expires at the same time as the session lifetime.

It should be noted that the session is deleted when the session lifetime has been met. Assuming that revocation is enabled, if the CRL check passes, the new session will be established. There will be loss of secure connectivity from the time the session is deleted to the time that the new session is created.

The lifetime of the session can be controlled locally by the IKEv2 authorization policy; within this policy, the following command can be used to set the lifetime of the created IKEv2 SA.

Click here to view code image

```
session-lifetime <120- 25920000>
```

The session lifetime can be set, using the IETF RADIUS attribute (27) Session-Timeout. This can be included in the RADIUS Access-Accept when a peer is authenticated, and the created IKEv2 SA be created for the configured lifetime.

The value can be any number with the 32-bit unsigned integer field, which denotes the lifetime in seconds and is configurable.

If an IKEv2 session is created which uses the session lifetime feature, this can be seen by using the **show crypto session detail** command.

The following example illustrates the establishment of two crypto sessions. The session using virtual-access 1 has a static lifetime with a little over five minutes remaining. When this timer expires, the session will be deleted.

Click here to view code image

```
Router#show crypto session
Crypto session current status

Interface: Virtual-Access1
Profile: default
Session status: UP-ACTIVE
Peer: 10.10.50.5 port 500
  Session ID: 3
  Session Lifetime: 00:05:22
  IKEv2 SA: local 10.10.10.1/500 remote 10.10.50.5/500 Active
  IPSEC FLOW: permit 47 host 10.10.10.1 host 10.10.50.5
        Active SAs: 2, origin: crypto-map

Interface: Virtual-Access2
Profile: default
Session status: UP-ACTIVE
Peer: 10.10.60.6 port 500
  Session ID: 6
  IKEv2 SA: local 10.10.10.1/500 remote 10.10.60.6/500 Active
  IPSEC FLOW: permit 47 host 10.10.10.1 host 10.10.60.6
```

```
        Active SAs: 2, origin: crypto-map
```

If IKEv2 debugging is enabled, using the command **debug crypto ikev2**, the adjusted IKEv2 lifetime can be seen within the generate syslog messages. The following illustrates a session that is established, where the session lifetime is set to 600 seconds, which then sets the IKEv2 lifetime to the same value.

Click here to view code image

```
IKEv2:Adjusting lifetime due to Session expiry time
IKEv2:(SESSION ID = 7,SA ID = 1):IKEV2 SA created; inserting SA into database. SA
lifetime timer (600 sec) started
%IKEV2-5-SA_UP: SA UP
```

When the session lifetime expires and the session is deleted, the following syslog message is generated in this instance.

Click here to view code image

```
(SESSION ID = 11,SA ID = 1):Queuing IKE SA delete request reason: session lifetime
expired
```

Summary

This chapter introduced a number of advanced IKEv2 topics. On Cisco IOS, IKEv2 fragmentation supports both the Cisco proprietary method and the IETF standardized method.

When using Cisco security group tagging, IPsec can be used to transport the SGT tag. IKEv2 performs the negotiation of this feature.

When using certificates for authentication, options exist for designing an architecture that will allow certificates that have been revoked or expired to delete the corresponding IKEv2 SAs. The IKEv2 session lifetime can be used to enable VPN sessions to exist for a defined value of time. This allows systems to be designed where the cryptographic protection and access is provided for the duration that it is needed.

References

https://tools.ietf.org/html/rfc1191 Path MTU Discovery

https://tools.ietf.org/html/rfc5280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

Chapter 7. IKEv2 Deployments

This chapter introduces a number of designs where IKEv2 is used. Each design will use a simple deployment of two routers with the focus on the configuration of IKEv2. Although each scenario uses only two routers, the configuration can scale as required if needed.

The configuration is intended to be as simple as possible, and the emphasis is focused on the IKEv2 configuration.

Pre-shared-key Authentication with Smart Defaults

This configuration is the simplest to set up. By using smart defaults, a VPN is created between two peers using minimal configuration: only the IKEv2 profile and corresponding IKEv2 keyring are required.

Figure 7-1 illustrates the topology. The transport network is using IPv6, and the overlay network is using IPv4.

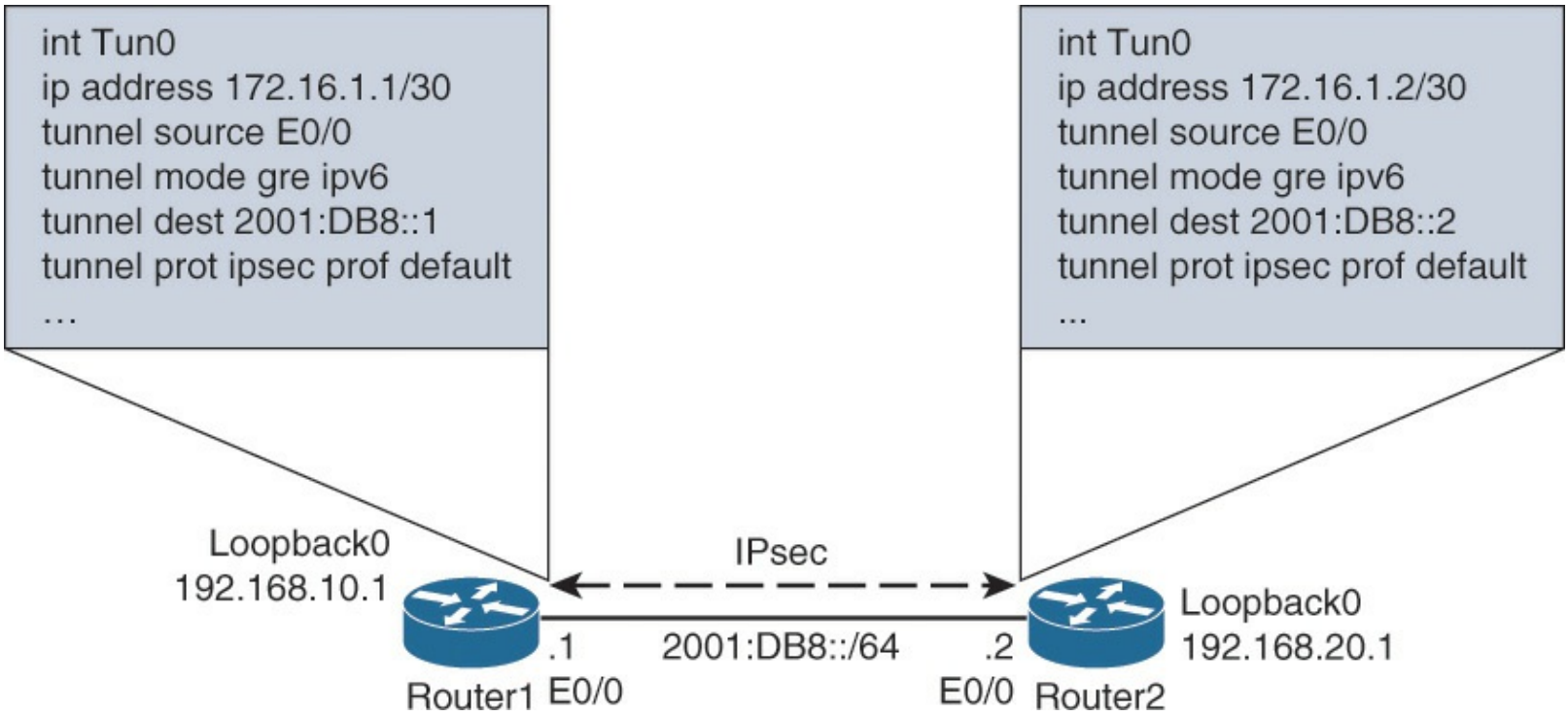


Figure 7-1 *PSK Authentication with Smart Defaults Topology*

The following example illustrates the relevant configuration used on Router1. This is a very minimal configuration which leaves little room for error.

Note that the shared secrets used in the example below are for illustrative purposes and, if used in a production environment, should contain sufficient entropy.

The example might seem complex as this scenario uses IPv4 and IPv6; however, the main focus of interest is to illustrate the IKEv2 configuration and the simplicity of using smart defaults.

An IKEv2 keyring is created with a peer entry which matches the peer's IPv6 address. Asymmetric pre-shared-keys are used with each device having a unique local and remote key.

Click here to view code image

```
crypto ikev2 keyring local_keyring
 peer 2001:DB8::2
```

```
 address 2001:DB8::2/128
 pre-shared-key local bartlett
 pre-shared-key remote inamdar
```

The IKEv2 profile is the mandatory component and matches the remote IPv6 address configured on Router2. The local IKEv2 identity is set to the IPv6 address configured on E0/0. The authentication is set to pre-shared-key with the locally configured keyring defined previously.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote address 2001:DB8::2/128
 identity local address 2001:DB8::1
 authentication remote pre-share
 authentication local pre-share
 keyring local local_keyring
```

The local loopback interface is configured, which will allow testing over the IPsec Security Association.

Click here to view code image

```
interface Loopback0
 ip address 192.168.10.1 255.255.255.0
```

The tunnel interface is created as tunnel mode GRE IPv6. This is required as the transport network is IPv6 and the overlay is IPv4. The default IPsec profile is used to protect this interface; this uses the default IKEv2 profile which was configured earlier.

Click here to view code image

```
interface Tunnel0
 ip address 172.16.1.1 255.255.255.252
 tunnel source Ethernet0/0
 tunnel mode gre ipv6
 tunnel destination 2001:DB8::2
 tunnel protection ipsec profile default
```

The physical interface used as the tunnel source uses IPv6.

Click here to view code image

```
interface Ethernet0/0
 no ip address
 ipv6 address 2001:DB8::1/64
```

Enhanced interior gateway routing protocol (EIGRP) is used to establish a peer relationship over the tunnel interface and distribute the loopback prefix.

Click here to view code image

```
router eigrp 1
 network 172.16.1.0 0.0.0.3
 network 192.168.10.0
```

The following example illustrates the relevant configuration on Router2.

Click here to view code image

```
interface Loopback0
 ip address 192.168.20.1 255.255.255.0
```

```
interface Tunnel0
 ip address 172.16.1.2 255.255.255.252
 tunnel source Ethernet0/0
 tunnel mode gre ipv6
 tunnel destination 2001:DB8::1
 tunnel protection ipsec profile default

interface Ethernet0/0
 no ip address
 ipv6 address 2001:DB8::2/64

router eigrp 1
 network 172.16.1.0 0.0.0.3
 network 192.168.20.0
```

The following example illustrates the EIGRP neighbor relationship built over the tunnel interface. The prefix for IP address assigned to the loopback interface on Router2 is reachable via the protected tunnel.

Click here to view code image

```
Router1#show ip route 192.168.20.0
Routing entry for 192.168.20.0/24
  Known via "eigrp 1", distance 90, metric 27008000, type internal
  Redistributing via eigrp 1
  Last update from 172.16.1.2 on Tunnel0, 00:12:04 ago
  Routing Descriptor Blocks:
  * 172.16.1.2, from 172.16.1.2, 00:12:04 ago, via Tunnel0
      Route metric is 27008000, traffic share count is 1
      Total delay is 55000 microseconds, minimum bandwidth is 100 Kbit
      Reliability 255/255, minimum MTU 1418 bytes
      Loading 1/255, Hops 1
```

The following example illustrates the IKEv2 SA that is created. The IKEv2 SA is protected by the PRF and integrity algorithms using SHA512, encryption using AES-CBC-256, and Diffie-Hellman group 5, which are the most preferred algorithms within the IKEv2 default proposal. The authentication is performed using pre-shared-key.

Click here to view code image

```
Router1#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

 IPv6 Crypto IKEv2  SA

Tunnel-id    fvrf/ivrf              Status
1          none/none            READY
Local  2001:DB8::1/500
Remote  2001:DB8::2/500
      Encr: AES-CBC, keysize: 256, PRF: SHA512, Hash: SHA512, DH Grp:5, Auth sign:
PSK, Auth verify: PSK
      Life/Active Time: 86400/10523 sec
      CE id: 1002, Session-id: 2
      Status Description: Negotiation done
      Local spi: 261B9BD2F208A02A       Remote spi: 0B28D2A21FC6304D
      Local id: 2001:DB8::1
      Remote id: 2001:DB8::2
      Local req msg id:  4              Remote req msg id:  4
      Local next msg id: 4              Remote next msg id: 4
```

```
      Local req queued:  4              Remote req queued:  4
      Local window:      5              Remote window:      5
...
```

The following example illustrates traffic being sent over the IPsec Security Association. The tunnel source and destination being the IPv6 addresses configured on the physical E0/0 interfaces.

Traffic is sent via the tunnel interface, from the locally configured loopback interface to the loopback on Router2.

Click here to view code image

```
Router1#ping 192.168.20.1 source 192.168.10.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.20.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.10.1

Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/5 ms
```

The IPsec Security Association is verified where the default IPsec transform set is used, which is created using Encapsulation Security Payload with AES-CBC-256 for encryption and SHA1-HMAC for integrity. Transport mode is used.

Click here to view code image

```
Router1#show crypto ipsec sa

interface: Tunnel0
    Crypto-map tag: Tunnel0-head-0, local addr 2001:DB8::1

   protected vrf: (none)
   local  ident (addr/mask/prot/port): (2001:DB8::1/128/47/0)
   remote ident (addr/mask/prot/port): (2001:DB8::2/128/47/0)
   current_peer 2001:DB8::2 port 500
     PERMIT, flags={origin_is_acl,}
    #pkts encaps: 523, #pkts encrypt: 523, #pkts digest: 523
    #pkts decaps: 523, #pkts decrypt: 523, #pkts verify: 523
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0
    #pkts not decompressed: 0, #pkts decompress failed: 0
    #send errors 0, #recv errors 0

     local crypto endpt.: 2001:DB8::1,
     remote crypto endpt.: 2001:DB8::2
     plaintext mtu 1462, path mtu 1500, ipv6 mtu 1500, ipv6 mtu idb Ethernet0/0
     current outbound spi: 0x5FC3C94A(1606666570)
     PFS (Y/N): N, DH group: none

     inbound esp sas:
      spi: 0xB8435B94(3091422100)
        transform: esp-aes esp-sha-hmac ,
        in use settings ={Transport, }
        conn id: 10, flow_id: SW:10, sibling_flags 80000001, crypto-map:
Tunnel0-head-0
        sa timing: remaining key lifetime (k/sec): (4315844/2543)
        IV size: 16 bytes
        replay detection support: Y
        Status: ACTIVE(ACTIVE)

     inbound ah sas:
```

```
      inbound pcp sas:

      outbound esp sas:
       spi: 0x5FC3C94A(1606666570)
         transform: esp-aes esp-sha-hmac ,
         in use settings ={Transport, }
         conn id: 9, flow_id: SW:9, sibling_flags 80000001, crypto-map:
Tunnel0-head-0
         sa timing: remaining key lifetime (k/sec): (4315844/2543)
         IV size: 16 bytes
         replay detection support: Y
         Status: ACTIVE(ACTIVE)
```

Elliptic Curve Digital Signature Algorithm Authentication

The scenario looks to use digital signatures to authenticate both peers. Rather than the more common RSA certificates, Elliptic Curve (EC) certificates are used that provide the ability to authenticate both parties, using the Elliptic Curve Digital Signature Algorithm (ECDSA).

The configuration in this example is intended to be simple, with the main focus on the IKEv2 configuration.

Figure 7-2 illustrates the physical IP addressing and the setup of the tunnel interface.

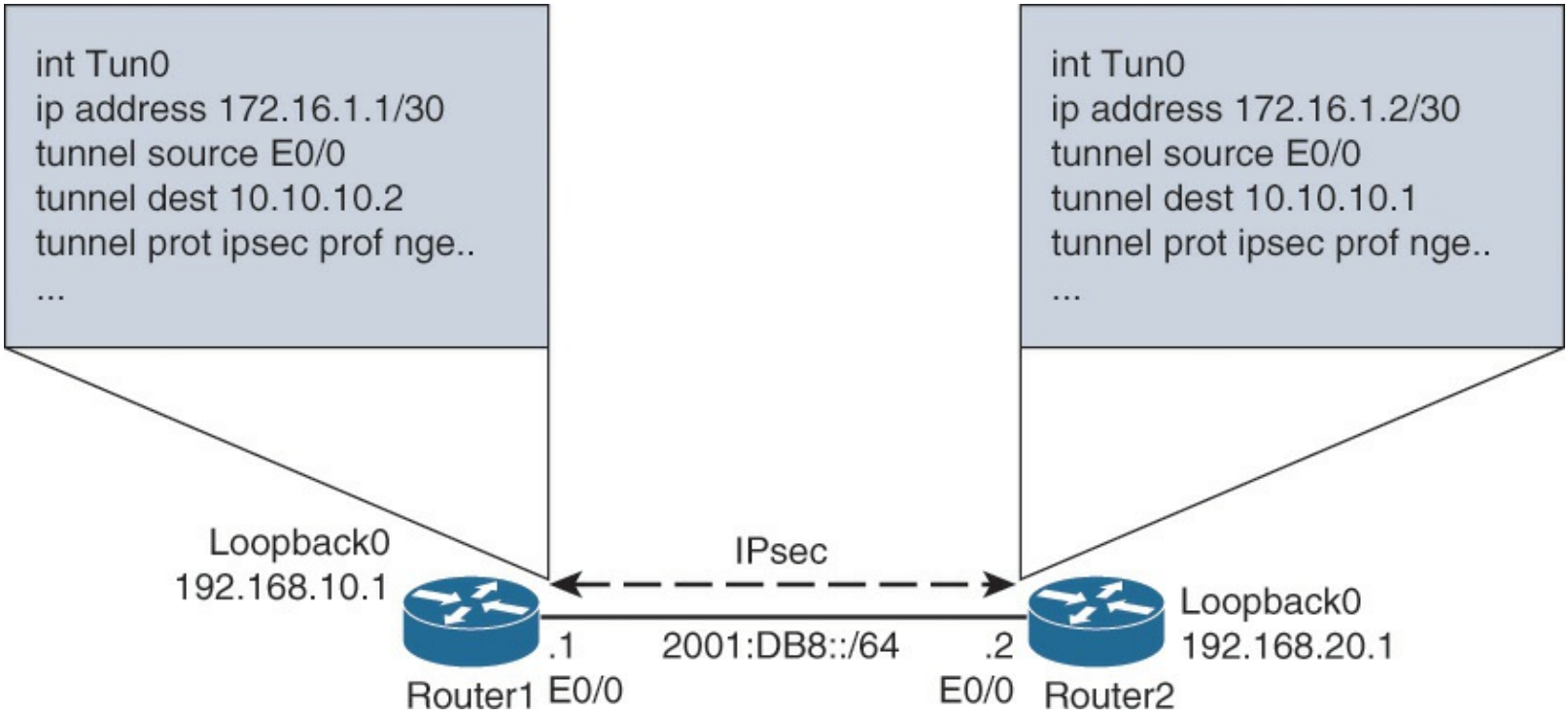


Figure 7-2 *Topology with Configuration*

In addition to ECDSA for authentication, Cisco Next Generation Encryption (NGE) algorithms secure the IKEv2 and IPsec session, as shown in Table 7-1.

Method	Algorithm
IKEv2 encryption	AES-GCM-256
IKEv2 PRF	SHA512
Diffie-Hellman	Group 21
Authentication	Elliptic Curve Digital Signature Algorithm
IPsec encryption	AES-GCM-256
IPsec PFS	Group 21

Table 7-1 *Security Algorithms Used*

Rather than using the default IKEv2 proposal, the default IKEv2 proposal is disabled, and a new IKEv2 proposal created containing the IKEv2 algorithms defined in Table 7-1.

Static routes are used to send traffic down the freshly created tunnel interface.

The following example illustrates the configuration that is used on Router1.

> **Note**
>
> Although not shown, the trustpoint uses a locally configured elliptic curve keypair.

The trustpoint is configured using manual enrollment, with the local and CA certificate.

Click here to view code image

```
crypto pki trustpoint ecdsa_tp
 enrollment terminal

crypto pki certificate chain ecdsa_tp
 certificate 6156E3D5000000000009
  308203BF 30820365 A0030201 02020A61 56E3D500 00000000 09300A06 082A8648
...
  68656c6c 6f736861 627333E4FDDC 642DA416 F57D4962 C5DF6545 FEC931FA F84BAF40
  A9829E
      quit
 certificate ca 780887F0CDD97E9E49DB893FA5D74238
  30820206 308201AB A0030201 02021078 0887F0CD D97E9E49 DB893FA5 D7423830
  0A06082A 8648CE3D 04030230 4F311330 11060A09 92268993 F22C6401 19160363
...
  816AA443 9191FBAC 731C
      quit
```

A certificate map is created that will match certificates containing a subject name of *cisco.com*. This is used within the IKEv2 profile to anchor the certificates presented by the peers. As this is a site-to-site VPN with only two peers, the certificate map could have been more granular to include the peer DN.

Click here to view code image

```
crypto pki certificate map certmap 10
 subject-name co cisco.com
```

The default IKEv2 proposal is disabled, and a new IKEv2 proposal is created that contains the

relevant cryptographic algorithms.

> **Note**
>
> Because this is a combined mode cipher, no integrity algorithm is required.

Click here to view code image

```
no crypto ikev2 proposal default
crypto ikev2 proposal nge
 encryption aes-gcm-256
 prf sha512
 group 21
```

An IKEv2 policy is created, which encompasses the IKEv2 proposal created above. Because the default IKEv2 proposal is disabled, this then ensures that only the IKEv2 proposal named *nge* will be used and minimizes the chance of mis-configuration.

```
crypto ikev2 policy default
 match fvrf any
 proposal nge
```

An IKEv2 profile is created, which uses the certificate map created earlier. The identity is set to DN, which will use the DN from the certificate. The authentication method is set to ECDSA and the PKI trustpoint used which was configured earlier. This profile will only match peer certificates, which contain the string *cisco.com* within the subject name. Dead-peer detection is enabled to ensure that the IKEv2 SA and corresponding IPsec Security Associations are torn down in a timely manner if IKE connectivity is lost.

Click here to view code image

```
crypto ikev2 profile nge
 match certificate certmap
 identity local dn
 authentication remote ecdsa-sig
 authentication local ecdsa-sig
 pki trustpoint ecdsa_tp
 dpd 10 2 on-demand
```

An IPsec transform set is created, which uses AES-GCM-256. Because this is a combined mode cipher, no integrity algorithm is required.

Click here to view code image

```
crypto ipsec transform-set nge-transform esp-gcm 256
 mode transport
```

The default IPsec profile is disabled, which ensures that it is not used due to mis-configuration. A new IPsec profile is created which uses the IKEv2 profile and IPsec transform-set created earlier. Additionally, perfect forward secrecy is enabled to ensure that a fresh Diffie-Hellman exchange is performed on rekey.

Click here to view code image

```
no crypto ipsec profile default

crypto ipsec profile nge-profile
```

```
 set transform-set nge-transform
 set pfs group21
 set ikev2-profile nge
```

A loopback interface is used that will allow traffic to be sourced from and destined to as it transverses the VPN.

Click here to view code image

```
interface Loopback0
 ip address 192.168.10.1 255.255.255.0
```

The tunnel interface is created with the relevant source interface configured and with the destination address of Router2. This is protected by the IPsec profile created above.

Click here to view code image

```
interface Tunnel0
 ip address 172.16.1.1 255.255.255.252
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.2
 tunnel protection ipsec profile nge-profile
```

The E0/0 interface is used as the tunnel source.

Click here to view code image

```
interface Ethernet0/0
 ip address 10.10.10.1 255.255.255.0
```

A static route is configured to send all traffic for the 192.168.20.0/24 network, which is the subnet protected by the peer, via the peer tunnel IP address.

Click here to view code image

```
ip route 192.168.20.0 255.255.255.0 172.16.1.2
```

Router2 has a nearly similar configuration; the following example illustrates the unique configuration. The tunnel interface has a unique IP address, and the destination is configured as E0/0 on Router1.

Note the unique IP address and the tunnel destination of Router1.

Click here to view code image

```
interface Tunnel0
 ip address 172.16.1.2 255.255.255.252
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.1
 tunnel protection ipsec profile nge-profile

interface Ethernet0/0
 ip address 10.10.10.2 255.255.255.0
```

The following example illustrates verification that the IKEv2 SA established. The algorithms used to secure the IKE session as described in Table 7-1 can be seen.

Click here to view code image

```
Router1#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
```

```
1         10.10.10.1/500        10.10.10.2/500        none/none             READY
      Encr: AES-GCM, keysize: 256, PRF: SHA512, Hash: None, DH Grp:21, Auth sign:
ECDSA, Auth verify: ECDSA
      Life/Active Time: 86400/6 sec
      CE id: 1030, Session-id: 13
      Status Description: Negotiation done
      Local spi: 313404E23B3A5707       Remote spi: 13FE5BCC09FFAAAB
      Local id: hostname=Router1.cisco.com
      Remote id: hostname=Router2.cisco.com
      Local req msg id:  2              Remote req msg id:  0
      Local next msg id: 2              Remote next msg id: 0
      Local req queued:  2              Remote req queued:  0
      Local window:      5              Remote window:      5
      DPD configured for 10 seconds, retry 2
      Fragmentation not configured.
      Extended Authentication not configured.
      NAT-T is not detected
      Cisco Trust Security SGT is disabled
      Initiator of SA : Yes

 IPv6 Crypto IKEv2  SA
```

The creation of the IPsec Security Association can be seen in the following example. The tunnel interface is configured with the default GRE mode, the traffic selectors can be seen indicating this by the use of IP protocol 47.

Click here to view code image

```
Router1#show crypto sockets

Number of Crypto Socket connections 1

   Tu0 Peers (local/remote): 10.10.10.1/10.10.10.2
       Local Ident  (addr/mask/port/prot): (10.10.10.1/255.255.255.255/0/47)
       Remote Ident (addr/mask/port/prot): (10.10.10.2/255.255.255.255/0/47)
       IPSec Profile: "nge-profile"
       Socket State: Open
       Client: "TUNNEL SEC" (Client State: Active)
Crypto Sockets in Listen state:
Client: "TUNNEL SEC" Profile: "nge-profile" Map-name: "Tunnel0-head-0"
```

The following example illustrates the route to 192.168.20.0/24, which be seen via the tunnel interface. All traffic intended for this network will be sent via the tunnel and encrypted by the corresponding IPsec Security Association.

Click here to view code image

```
Router1#show ip route 192.168.20.0 255.255.255.0
Routing entry for 192.168.20.0/24
  Known via "static", distance 1, metric 0
  Routing Descriptor Blocks:
  * 172.16.1.2
      Route metric is 0, traffic share count is 1
Router1#show ip route 172.16.1.2
Routing entry for 172.16.1.0/30
  Known via "connected", distance 0, metric 0 (connected, via interface)
  Routing Descriptor Blocks:
  * directly connected, via Tunnel0
      Route metric is 0, traffic share count is 1
```

Traffic is sent from Router1 to Router2 via the tunnel interface. Note that this traffic has been protected by the IPsec Security Association, as indicated by the increasing *encaps* and *decaps* counters.

Click here to view code image

```
Router1#ping 192.168.20.1 source 192.168.10.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.20.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.10.1

Success rate is 100 percent (5/5), round-trip min/avg/max = 3/4/6 ms

Router1#show crypto ipsec sa

interface: Tunnel0
    Crypto-map tag: Tunnel0-head-0, local addr 10.10.10.1

   protected vrf: (none)
   local  ident (addr/mask/prot/port): (10.10.10.1/255.255.255.255/47/0)
   remote ident (addr/mask/prot/port): (10.10.10.2/255.255.255.255/47/0)
   current_peer 10.10.10.2 port 500
     PERMIT, flags={origin_is_acl,}
    #pkts encaps: 10, #pkts encrypt: 10, #pkts digest: 10
    #pkts decaps: 10, #pkts decrypt: 10, #pkts verify: 10
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0
    #pkts not decompressed: 0, #pkts decompress failed: 0
    #send errors 0, #recv errors 0

     local crypto endpt.: 10.10.10.1, remote crypto endpt.: 10.10.10.2
     plaintext mtu 1466, path mtu 1500, ip mtu 1500, ip mtu idb Ethernet0/0
     current outbound spi: 0x3FD4A2AF(1070899887)
     PFS (Y/N): Y, DH group: group21

     inbound esp sas:
      spi: 0x349334C6(882062534)
        transform: esp-gcm 256 ,
        in use settings ={Transport, }
        conn id: 6, flow_id: SW:6, sibling_flags 80000000, crypto-map:
Tunnel0-head-0
        sa timing: remaining key lifetime (k/sec): (4207250/3566)
        IV size: 8 bytes
        replay detection support: Y
        Status: ACTIVE(ACTIVE)

     inbound ah sas:

     inbound pcp sas:

     outbound esp sas:
      spi: 0x3FD4A2AF(1070899887)
        transform: esp-gcm 256 ,
        in use settings ={Transport, }
        conn id: 5, flow_id: SW:5, sibling_flags 80000000, crypto-map:
Tunnel0-head-0
        sa timing: remaining key lifetime (k/sec): (4207250/3566)
        IV size: 8 bytes
        replay detection support: Y
        Status: ACTIVE(ACTIVE)
```

```
outbound ah sas:

outbound pcp sas:
```

RSA Authentication Using HTTP URL Lookup

In this scenario, we will use RSA certificates to authenticate both peers. However, for Router2, we will not send the certificate within the IKE AUTH exchange, but will send a HTTP URL from Router2 to Router1 to inform it where to obtain the certificate. Router1 will then retrieve the certificate from the HTTP URL and verify that the presented AUTH payload was signed by the private key relating to the public key contained within the certificate.

Router1 has been set up as a certificate authority; from this CA, a certificate is obtained for both Router1 and Router2. These certificates are used to authenticate the IKEv2 SA.

Figure 7-3 illustrates the operation of the HTTP URL lookup feature. Router2 will sign the AUTH payload with its private key. Router1 will retrieve the certificate from the HTTP server and validate the AUTH payload by using the public key obtained from the retrieved certificate.

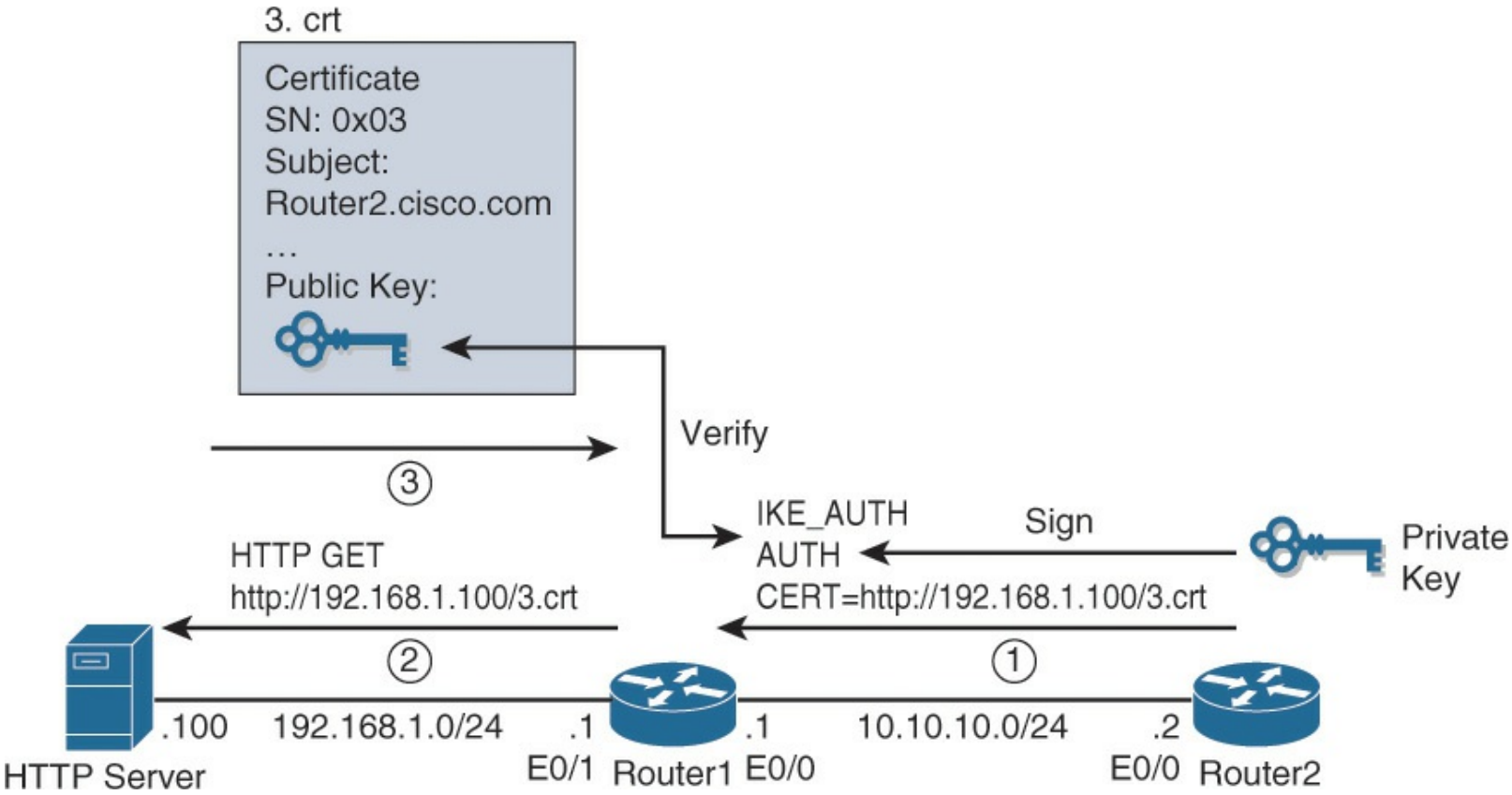


Figure 7-3 *HTTP URL Lookup Feature*

> **Note**
>
> The certificate generated by the IOS CA is in Privacy Enhanced Mail (PEM) format. Although the IKEv2 RFC states that the HASH and URL feature returns a URL with the SHA1 hash of the requested certificate, Cisco IOS allows for any URL to be used. As per the IKEv2 RFC, Cisco IOS requires the obtained certificate to be in distinguished encoding rules (DER) encoding. The following example illustrates the OpenSSL commands to manually convert a certificate from PEM to DER encoding, with the PEM encoded certificate in file 3.crt.

Click here to view code image

```
openssl x509 -outform der -in 3.crt -out 3.der
```

Figure 7-4 illustrates the topology used in the tunnel interface configuration.

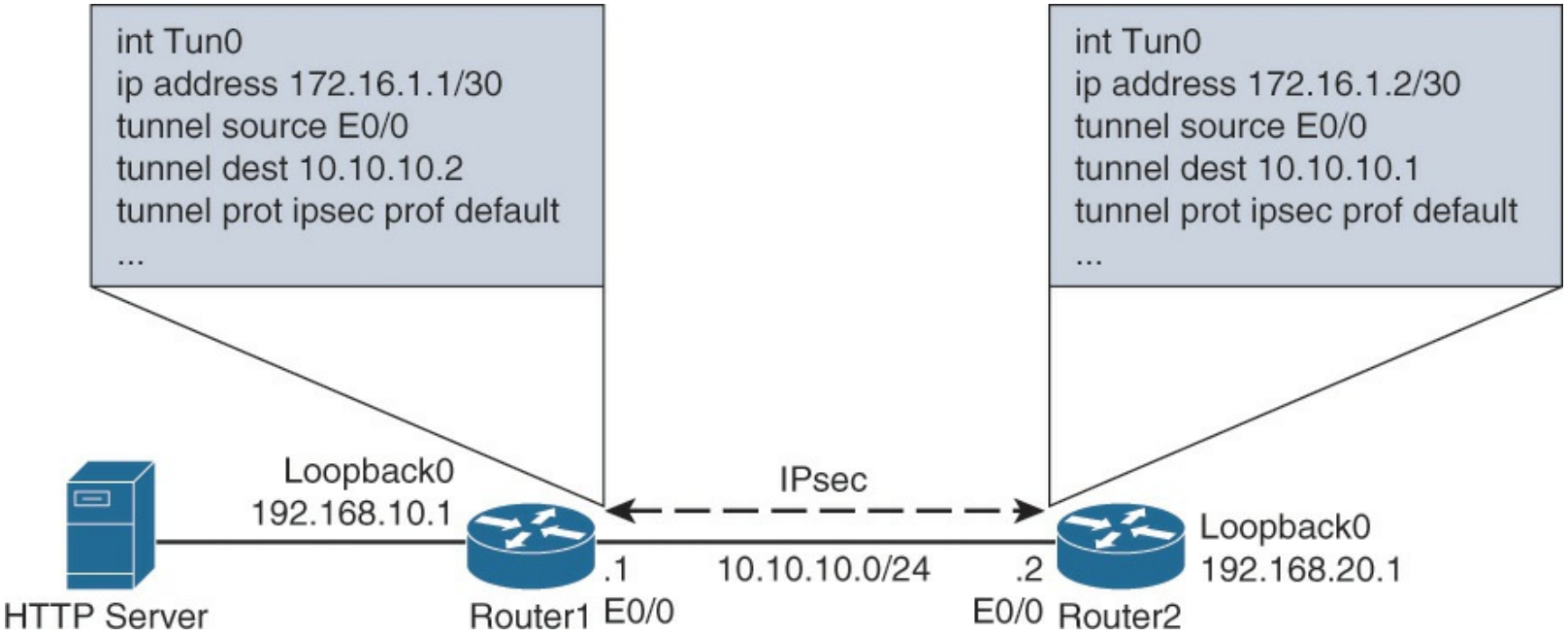


Figure 7-4 *Topology with Configuration*

The configuration is similar to the ECDSA example earlier, but RSA certificates are used, which results in a different authentication method. However, the base concepts are the same with regards to the PKI.

The subject information access (SIA) is an attribute within a certificate that defines some type of offered services. An example of where to access a server can be included in the SIA with a uniform resource identifier (URI). The SIA is amended to contain the URL that the peer will use for the HTTP URL lookup. This is achieved by the use of the certificate map that matches the locally used certificate and is attached to the trustpoint. This removes the inclusion of the certificate within the IKE exchange and uses the value defined in the SIA as the location for the peer to obtain the certificate.

The following example illustrates the configuration used on Router2.

The PKI trustpoint is defined; it has been authenticated, and the local device enrolled. The critical component to ensure that this client does not send its certificate but instead sends the HTTP URL is the *match certificate* command. This command will match the defined certificate map and override the SIA to contain the configured URL. This is then sent in replacement of the certificate in the IKE_AUTH exchange.

Click here to view code image

```
crypto pki trustpoint CA
 enrollment url http://10.10.10.1:80
 revocation-check crl
 match certificate local override sia 1 http://192.168.1.100/3.der
```

A certificate map is created that will match certificates containing a subject name of *router1.cisco.com*. This is used within the IKEv2 profile to anchor the peer's presented certificate.

Click here to view code image

```
crypto pki certificate map certmap 10
 subject-name eq router1.cisco.com
```

The following certificate map is used by the match statement within the trustpoint configuration to match the local certificate. This is achieved by matching the local subject name (which is not case sensitive) of *router2*.

Click here to view code image

```
crypto pki certificate map local 10
 subject-name co router2
```

The mandatory IKEv2 profile is configured which uses the certificate map created earlier. This will match any certificates which contain a subject name of *cisco.com*. The authentication method is set to RSA signatures, and the trustpoint configured earlier is used.

Click here to view code image

```
crypto ikev2 profile default
 match certificate certmap
 identity local dn
 authentication remote rsa-sig
 authentication local rsa-sig
 pki trustpoint CA
```

The tunnel interface is created with the relevant source interface configured and the destination address of Router1. This is protected by the default IPsec profile which uses the default IKEv2 profile which was created earlier.

Click here to view code image

```
interface Tunnel0
 ip address 172.16.1.2 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.1
 tunnel protection ipsec profile default
```

The following physical interface is used as the tunnel source.

Click here to view code image

```
interface Ethernet0/0
 ip address 10.10.10.2 255.255.255.0
```

Note

Should a certificate hierarchy exist where there is a requirement to send a certificate chain with multiple URLs in multiple CERT payloads starting from "ID cert url," "subca1," "subca2," until "root CA"; then each additional certificate can be included as a separate line within the trustpoint configuration as illustrated below.

Click here to view code image

```
crypto pki trustpoint CA
 match certificate local override sia 1 http://192.168.1.100/3.der
 match certificate local override sia 2 http://192.168.1.100/subca1.der
 match certificate local override sia 3 http://192.168.1.100/subca2.der
 match certificate local override sia 4 http://192.168.1.100/root.der
```

The following example illustrates the configuration used on Router1.

The certificate authority function is enabled. Note that the automatic granting of certificates is used here for ease of configuration and should not occur in a production environment where un-authenticated access to the CA can occur.

```
crypto pki server local
 database level complete
 no database archive
 grant auto
```

The relating PKI trustpoint for the IOS CA is:

```
crypto pki trustpoint local
 revocation-check crl
 rsakeypair local
```

A trustpoint is used to enroll into the local CA.

Click here to view code image

```
crypto pki trustpoint CA
 enrollment url http://10.10.10.1:80
 revocation-check crl
```

A certificate map is created that will match certificates containing a subject name of *router2.cisco.com*. This is used within the IKEv2 profile to anchor the peer's presented certificate.

Click here to view code image

```
crypto pki certificate map certmap 10
 subject-name eq router2.cisco.com
```

The mandatory IKEv2 profile is configured that uses the certificate map created earlier. This will match any certificates, which contain a subject name of *cisco.com*. The authentication method is set to RSA signatures, and the trustpoint configured earlier is used.

Click here to view code image

```
crypto ikev2 profile default
 match certificate certmap
 identity local dn
 authentication remote rsa-sig
 authentication local rsa-sig
 pki trustpoint CA
```

The tunnel interface is created with the relevant source interface configured, and the destination address of Router1. This is protected by the default IPsec profile that uses the default IKEv2 profile, which was created earlier.

Click here to view code image

```
interface Tunnel1
 ip address 172.16.1.1 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.2
 tunnel protection ipsec profile default
```

The physical interface used as the tunnel source.

Click here to view code image

```
interface Ethernet0/0
 ip address 10.10.10.1 255.255.255.0
```

The physical interface used to reach the HTTP server containing the certificates.

Click here to view code image

```
interface Ethernet0/1
 ip address 192.168.1.1 255.255.255.0
```

> **Note**
>
> When using the HTTP URL lookup feature, the router that retrieves the HTTP URL should be protected from malicious intent by restricting HTTP access to only the server storing the certificates. As the certificate obtained via the HTTL URL method is processed prior to authentication, an intruder could redirect the gateway to a large file containing garbage, or a URI that will slowly introduce a file, a little at a time, causing a DoS on the gateway. Mitigation can be achieved using controls, such as access-control-lists, control-plane policing, or control-plane protection.

The following example illustrates IKEv2 debugs taken from Router1. It can be seen that Router2 sends the IKE_AUTH exchange with the CERT payload containing the HASH and URL format. Also note the NOTIFY payload which indicates the HTTP URL method is supported.

Click here to view code image

```
IKEv2:(SESSION ID = 4,SA ID = 1):Received Packet [From 10.10.10.2:500/To
10.10.10.1:500/VRF i0:f0]
Initiator SPI : 52D538043A8E330C - Responder SPI : 5CE063D07E8745EA Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST
IKEv2-PAK:(SESSION ID = 4,SA ID = 1):Next payload: ENCR, version: 2.0 Exchange
type: IKE_AUTH, flags: INITIATOR Message id: 1, length: 816
Payload contents:
 VID  Next payload: IDi, reserved: 0x0, length: 20
 IDi  Next payload: CERT, reserved: 0x0, length: 44
    Id type: DER ASN1 DN, Reserved: 0x0 0x0
 CERT  Next payload: CERTREQ, reserved: 0x0, length: 52
    Cert encoding Hash and URL of PKIX
 CERTREQ  Next payload: NOTIFY, reserved: 0x0, length: 25
    Cert encoding Hash and URL of PKIX
 NOTIFY(HTTP_CERT_LOOKUP_SUPPORTED)  Next payload: AUTH, reserved: 0x0, length: 8
    Security protocol id: Unknown - 0, spi size: 0, type:
HTTP_CERT_LOOKUP_SUPPORTED
 AUTH  Next payload: CFG, reserved: 0x0, length: 136
    Auth method RSA, reserved: 0x0, reserved 0x0
 CFG  Next payload: SA, reserved: 0x0, length: 304
    cfg type: CFG_REQUEST, reserved: 0x0, reserved: 0x0
```

A short time later, Router1 opens a TCP socket with 192.168.1.100, when the certificate is obtained.

Click here to view code image

```
TCP: sending SYN, seq 2191097267, ack 0
TCP0: Connection to 192.168.1.100:80, advertising MSS 1460
TCP0: state was CLOSED -> SYNSENT [42603 -> 192.168.1.100(80)]
TCP0: state was SYNSENT -> ESTAB [42603 -> 192.168.1.100(80)]
```

```
TCP: tcb 32417230 connection to 192.168.1.100:80, peer MSS 1460, MSS is 1460
```

The following example illustrates verification on Router1 that the certificate was obtained by way of HTTP.

Click here to view code image

```
Router1#show crypto ikev2 stats ext-service
-------------------------------------------------------------
AAA OPERATION                                PASSED     FAILED
-------------------------------------------------------------
RECEIVING PSKEY                                 0          0
AUTHENTICATION USING EAP                        0          0
START ACCOUNTING                                0          0
STOP ACCOUNTING                                 0          0
AUTHORIZATION                                   0          0
-------------------------------------------------------------
IPSEC OPERATION                              PASSED     FAILED
-------------------------------------------------------------
IPSEC POLICY VERIFICATION                       3          0
SA CREATION                                     3          0
SA DELETION                                     3          0
-------------------------------------------------------------
CRYPTO ENGINE OPERATION                      PASSED     FAILED
-------------------------------------------------------------
DH PUBKEY GENERATED                            34          0
DH SHARED SECKEY GENERATED                     29          0
SIGNATURE SIGN                                 28          0
SIGNATURE VERIFY                                3          0
-------------------------------------------------------------
PKI OPERATION                                PASSED     FAILED
-------------------------------------------------------------
VERIFY CERTIFICATE                              3          0
FETCHING CERTIFICATE USING HTTP                 1          0
FETCHING PEER CERTIFICATE USING HTTP            1          0
GET ISSUERS                                    31          0
GET CERTIFICATES FROM ISSUERS                  28          0
GET DN FROM CERT                                3          0
-------------------------------------------------------------
GKM OPERATION                                PASSED     FAILED
-------------------------------------------------------------
GET_POLICY                                      0          0
SET_POLICY                                      0          0
```

The certificate that is obtained via HTTP is cached locally. By default, 200 certificates will be cached. As the certificate is cached, if the IKE session drops and is re-established, the certificate will not be required to be obtained via HTTP as it is already cached. This saves numerous HTTP requests to occur if the peer is required to re-authenticate. The following example illustrates viewing the contents of the certificate cache.

Click here to view code image

```
Router1#show crypto ikev2 certificate-cache
No of entries in ikev2 certificate-cache = 1

Certificate entry:
Certificate
  Status: Available
  Certificate Serial Number (hex): 03
  Certificate Usage: General Purpose
```

```
  Issuer:
    cn=CA.cisco.com
  Subject:
    Name: Router2.cisco.com
    hostname=Router2.cisco.com
  Validity Date:
    start date: 10:44:26 UTC Feb 8 2016
    end   date: 10:44:26 UTC Feb 7 2017
  Associated Trustpoints:
```

The following example illustrates the IKEv2 SA being verified. The cryptographic algorithms used have been negotiated via the use of smart defaults. The authentication method of RSA can be seen. There is no differentiation that the certificate was received via the HTTP URL method; the authentication is performed in the same manner as RSA authentication when certificates are sent in the IKE_AUTH exchange.

Click here to view code image

```
Router1#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         10.10.10.1/500        10.10.10.2/500        none/none            READY
      Encr: AES-CBC, keysize: 256, PRF: SHA512, Hash: SHA512, DH Grp:5, Auth sign:
RSA, Auth verify: RSA
      Life/Active Time: 86400/509 sec
      CE id: 1034, Session-id: 7
      Status Description: Negotiation done
      Local spi: 5CE063D07E8745EA       Remote spi: 52D538043A8E330C
      Local id: hostname=Router1.cisco.com
      Remote id: hostname=Router2.cisco.com
      Local req msg id:  0              Remote req msg id:  2
      Local next msg id: 0              Remote next msg id: 2
      Local req queued:  0              Remote req queued:  2
      Local window:      5              Remote window:      5
      DPD configured for 0 seconds, retry 0
      Fragmentation not  configured.
      Extended Authentication not configured.
      NAT-T is not detected
      Cisco Trust Security SGT is disabled
      Initiator of SA : No
```

IKEv2 Cookie Challenge and Call Admission Control

The following scenario highlights the use of the cookie challenge and the maximum in negotiation SA features, and the benefits that each brings.

IKEv2 call admission control (CAC) limits the maximum number of IKEv2 SAs that can be established. CAC limits the number of simultaneous negotiations with the default being 40 in-negotiation SAs, although this value is configurable using the **crypto ikev2 limit max-in-negotation-sa** command.

To illustrate the CAC in action, the architecture in Figure 7-5 was developed. This setup consists of an IOS device acting as a VPN headend. Imagine a device created to send many IKE_SA_INIT requests to the headend from random spoofed source IP addresses. The IOS headend is configured with a default gateway, which is where all replies to any received IKE_SA_INIT messages will be sent and then discarded. The IKEv2 generator is pre-configured with an IKEv2 proposal that will be

accepted by the IKEv2 headend and sends approximately 12 spoofed packets every second.

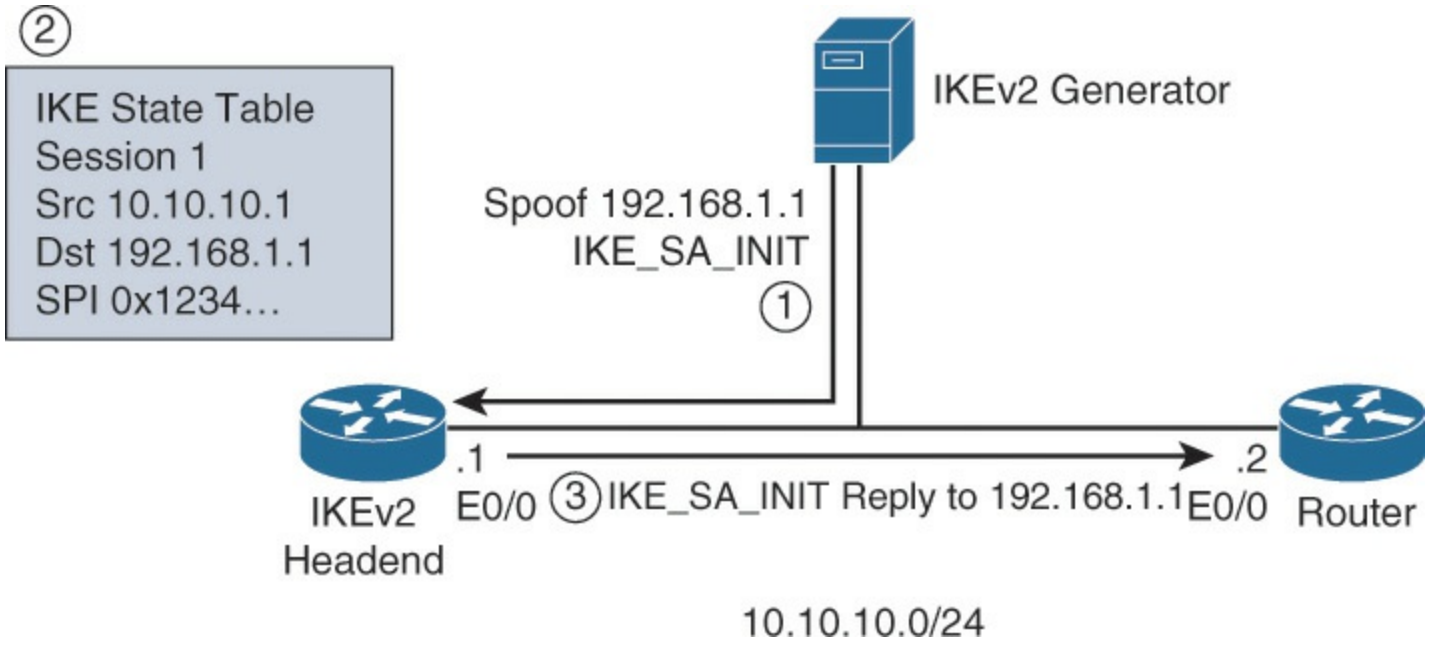


Figure 7-5 *CAC Architecture*

The IKEv2 generator sends an IKE_SA_INIT request with a spoofed source IP address of 192.168.1.1 to 10.10.10.1. The IKEv2 headend receives the IKE_SA_INIT, checks that the transforms are valid, allocates state and returns its IKE_SA_INIT response. This response will be received by the router and then forwarded to the 192.168.1.1 destination where it will be discarded.

The hardware used for the IKEv2 headend was purposely chosen as a low-powered device. This was to illustrate the load when generating a large number Diffie-Hellman calculations and the software crypto engine was used. The following example illustrates the CPU history when a constant stream of spoofed IKE_SA_INIT requests is sent from the IKEv2 generator. The sudden initial spike in CPU (40 to 60 seconds) is due to the device processing the first forty spoofed IKE_SA_INIT requests, these are processed and replies sent. The CPU then drops to zero percent for approximately fifteen seconds and once again rises back to near full CPU at ninety percent. The drop in CPU processing was due to the CAC feature becoming active. Once forty IKE SAs are in negotiation, no more IKE_SA_INIT requests will be processed. Although the IKEv2 generator is sending a constant stream of these, the IKEv2 headend will only process forty at any given time (although this value is configurable). Some of the initial forty requests time out, and the state for these are removed before any new requests are processed and state allocated.

Click here to view code image

```
Router#show processes cpu history

  100
   90           *****
   80           *****                     *****
   70           *****                     *****
   60           *****                     *****
   50           **********                **********
   40           **********                **********
   30           **********                **********
   20           **********                **********
   10           **********              ***************
     0....5....1....1....2....2....3....3....4....4....5....5....6
                0    5    0    5    0    5    0    5    0    5    0
```

```
                CPU% per second (last 60 seconds)
```

When an IKEv2 device acting as a responder receives a number of half-open IKE_SA_INIT requests, the cookie challenge mechanism can be deployed. This will enable the responder to include the cookie notification payload in the response to the initiator. The responder does not allocate any state to the session. If the initiator was legitimate, the response containing the cookie will reach the initiator who will then re-attempt the IKE_SA_INIT exchange, including the cookie notification payload, which is then verified by the responder. The responder will then allocate state to the IKE session.

If a device is under a Denial-of-Service (DoS) attack where spoofed IKE_SA_INIT are sent with the purpose of overloading the CPU, the device can be configured to activate the cookie-challenge mechanism. In this situation, the responder will reply with the cookie notification payload. Because this reply is sent to an IP address that was spoofed by an attacker, this reply will be discarded, or dropped by the receiver.

To illustrate this behavior, the IKEv2 headend was amended to allow 1000 in negotiation SAs. The following example shows the command used to achieve this.

Click here to view code image

```
Router(config)#crypto ikev2 limit max-in-negotation-sa 1000
```

The CPU of the IKEv2 headend was then constantly at 100 percent. This was due to the amount of constant spoofed IKE_SA_INIT requests from the IKEv2 generator that overwhelmed the IKEv2 state machine.

To rectify this issue, the cookie-challenge is enabled by default. This was enabled, using the value of 0, so all received IKE_SA_INIT requests will be returned with the cookie notification payload.

Click here to view code image

```
Router(config)#crypto ikev2 cookie-challenge 0
```

The value configured can be between 0 and 1000, which denotes the maximum number of in-negotiation IKE SAs before the cookie challenge is engaged.

No state is allocated to any IKE sessions as all IKE_SA_INIT replies are resent. The following example illustrates the impact that enabling the cookie challenge mechanism has. Once cookie challenge is enabled, the CPU drops from 100 to 0 percent. This is due to the fact that no state is allocated to any of the received IKE_SA_INIT requests.

Click here to view code image

```
Router#show processes cpu history

  100       ************************************************************
   90       ************************************************************
   80       ************************************************************
   70       ************************************************************
   60       ************************************************************
   50       ************************************************************
   40       ************************************************************
   30       ************************************************************
   20       ************************************************************
   10       ************************************************************
     0....5....1....1....2....2....3....3....4....4....5....5....6
```

```
0    5    0    5    0    5    0    5    0    5    0
CPU% per second (last 60 seconds)
```

The cookie challenge is a useful feature when an IKEv2 headend is under a DoS attack whereby source IP addresses are spoofed. It can be enabled by default. However, this will incur an additional two-packet exchange to any IKE negotiation which might not be optimal in some situations. Using a value for the maximum in negotiation SAs that is a little higher than what is observed in a known good state will allow this mechanism to engage should a DoS condition occur.

Summary

The examples used in this chapter illustrate a variety of IKEv2 configurations. Numerous authentication methods were used to illustrate the broad range of options available and the benefits that they bring. Smart defaults were used to show the simplicity of the configuration when these are employed. PKI is mandatory when using RSA or EC digital signatures which isn't needed when using pre-shared-key authentication. However, this is not as scalable.

The use of the HTTP URL cert feature was described, where the certificate is not sent in the exchange but instead is retrieved by the IKEv2 peer. This allows for a substantially reduced packet size of the IKE_AUTH exchange.

The use of the maximum in-negotiation SAs and the cookie challenge mechanism was observed to illustrate how IKE can be susceptible to DoS attacks. The use of the cookie notification payload can reduce the impact of a DoS attack; however, in non-DoS conditions, it does add an additional round trip to any IKEv2 exchange.

Part V: FlexVPN

Chapter 8. Introduction to FlexVPN

This chapter introduces the Cisco IOS FlexVPN solution. The chapter discusses the FlexVPN building blocks, builds on the foundational IKEv2 configuration covered in the earlier chapters, and explains the concepts and configurations relevant to FlexVPN. The chapter lays the foundation for the FlexVPN server and FlexVPN client chapters.

The chapter covers the following main topics

- FlexVPN overview
- FlexVPN building blocks
- IKEv2 name mangler
- IKEv2 authorization policy
- FlexVPN authorization
- FlexVPN configuration exchange
- FlexVPN routing

FlexVPN Overview

FlexVPN is a robust, standards-based VPN technology that enables large businesses to securely connect branch offices and remote users. It provides significant cost savings compared to the cost of supporting multiple separate types of virtual private network (VPN) solutions such as GRE (generic routing encapsulation), crypto map, and virtual tunnel interface (VTI)-based solutions. FlexVPN relies on open-standards-based IKEv2 as a security technology and provides a number of Cisco-specific enhancements to provide high levels of security, added value, and competitive differentiations.

FlexVPN is supported on Cisco Integrated Services Router Generation 2 (ISR G2) platforms running IOS and Aggregation Services Routers (ASR 1000) running IOS-XE, including Cloud Services Router 1000v (CSR 1000v) and ISR4000 series routers. Cisco AnyConnect is supported as a client, in addition to any compatible third-party client.

The Rationale

Over the years, Cisco IPsec VPN solutions have evolved from crypto maps to tunnel protection as listed:

- Static crypto maps
- Dynamic crypto maps
- Legacy EzVPN (Easy VPN) based on crypto maps
- GRE interfaces with crypto maps on WAN interface
- GRE interfaces with tunnel protection
- IPsec VTI (virtual tunnel interface): static and dynamic VTI
- DMVPN (dynamic multipoint VPN)
- Enhanced EzVPN based on IPsec VTI

- GET VPN (group encrypted transport VPN)

Because each solution serves different purposes, those different features were developed relatively independently, each with their own configuration constructs, show commands, underlying interface infrastructure, and feature sets while sharing the basic building blocks of ISAKMP and IPsec. An example of this divergence is that while DMVPN predominantly uses multipoint GRE interfaces, supports dynamic overlay routing, and does not support AAA (authentication, authorization, and accounting) and per-user or per-peer policy out of the box, EzVPN uses point-to-point IPsec VTI interfaces, integrates with AAA, and supports per-user or per-peer policy but does not support dynamic overlay routing. This separation and divergence has several drawbacks, notably in terms of learning curve and ease of troubleshooting due to the various concepts and commands involved. It is difficult to configure the various IPsec VPN solutions on a single device.

The IKEv2 protocol brought together a lot of functionalities that were earlier defined in multiple RFCs. The introduction of IKEv2 into Cisco IOS was used as an opportunity to unify the different VPN solutions, bringing all solutions under the same configuration and basic building blocks, and making available all the protocol functionalities and features to all VPN topologies. This resulted in FlexVPN.

With the exception of GET VPN that essentially does not create a VPN but provides encryption services to an existing layer-3 VPN, such as MPLS (multiprotocol label switching), FlexVPN encompasses all previous IPsec VPN functionality. FlexVPN is supported by a vastly improved framework based on the IKEv2 standard, point-to-point tunnel interfaces, and the AAA framework.

FlexVPN Value Proposition

FlexVPN is an IKEv2-based unified Cisco IOS VPN solution that caters to multiple VPN topologies. The key value proposition of FlexVPN is the unified user interface, underlying infrastructure and features across the VPN topologies. Some of the benefits of FlexVPN are:

- Standards-based, interoperable with non-Cisco IKEv2 implementations
- Support for multiple VPN topologies such as point-to-point, remote-access, hub-spoke, and dynamic mesh
- Multiple VPN technologies under one command line interface (CLI)
- Reduction in training to learn multiple VPN technologies
- Unified configuration and show commands, underlying interface infrastructure, and features across the supported VPN topologies
- Support for per-user or per-peer policy
- Support for dynamic overlay routing
- Integration with Cisco IOS AAA infrastructure
- Support for GRE and native IPsec encapsulations with auto-detection of encapsulation protocol
- Support for IPv4 and IPv6 overlay and underlay with auto-detection of IP transport type

FlexVPN Building Blocks

This section discusses the main building blocks that FlexVPN is based upon

- IKEv2

- Cisco IOS point-to-point (P2P) tunnel interfaces
- Cisco IOS AAA infrastructure

IKEv2

FlexVPN is based on IKEv2 and hence brings all the IKEv2 protocol features such as configuration exchange, IKEv2 redirect for server load balancing, cookie challenge for DoS mitigation, NAT traversal, IKEv2 fragmentation, and Cisco IOS IKEv2 features, such as IKEv2 call admission control, session deletion on certificate expiry, or revocation to all the supported VPN topologies. FlexVPN makes use of the IKEv2 configuration exchange in order to exchange policy parameters between peers (typically between FlexVPN client and server) and to exchange routing information between peers. It adds routes to remote overlay addresses and protected subnets and serves as a lightweight overlay routing mechanism.

Cisco IOS Point-to-Point Tunnel Interfaces

FlexVPN uses per-peer point-to-point (P2P) tunnel interfaces for all the supported VPN topologies with either GRE or native IPsec encapsulation. GRE encapsulation offers the benefit of native support for IP dual stack because it can carry both IPv4 and IPv6 overlay traffic over either IPv4 or IPv6 transport, compared to native IPsec encapsulation that can carry either IPv4 or IPv6 overlay traffic over either IPv4 or IPv6 transport and is useful for interoperating with implementations that do not support GRE. The P2P tunnel interface is statically configured on FlexVPN initiators and is dynamically instantiated from a virtual-template interface on FlexVPN responders. Table 8-1 lists the point-to-point tunnel interface types used by FlexVPN.

P2P Tunnel Interface Type	Usage
Static GRE interface (**tunnel mode gre**)	Initiator
Static IPsec VTI (sVTI) (**tunnel mode IPsec**)	Initiator
Virtual-template interface of type tunnel with ■ GRE encapsulation (**tunnel mode IPsec**) ■ Native IPsec encapsulation (**tunnel mode IPsec**) ■ auto-detection of tunnel mode and transport protocol (IPv4/IPv6)	Responder
Virtual-access interface cloned from virtual-template	Responder

Table 8-1 *P2P Tunnel Interfaces Used by FlexVPN*

Configuring Static P2P Tunnel Interfaces

On initiators, the IPsec Security Associations (SAs) originate from an IPsec interface. With FlexVPN, the IPsec interface is a tunnel interface with **tunnel protection** applied. The IPsec parameters used in the SA negotiation are derived from the tunnel interface configuration and especially the tunnel encapsulation mode determines the IPsec Security Association traffic selectors. Table 8-2 shows the tunnel encapsulation modes and the corresponding outbound IPsec Security Association negotiation parameters on an initiator.

Tunnel Encapsulation Mode	Outbound IPsec Security Association parameters
tunnel mode gre	IPsec peer addresses: Local peer address = tunnel source address Remote peer address = tunnel destination address IPsec Security Association traffic selectors: Local address = tunnel source address Local mask = host mask Remote address = tunnel destination address Remote mask = host mask Protocol = GRE (47) Local/Remote ports = Not applicable Note: This is referred to as *GRE host-to-host SA*
tunnel mode IPsec	IPsec peer addresses: Local peer address = tunnel source address Remote peer address = tunnel destination address IPsec Security Association traffic selectors: Local address = any Local mask = any Remote address = any Remote mask = any Protocol = any Local/Remote ports = any Note: This is referred to as *any-to-any SA*

Table 8-2 *Tunnel Encapsulation Modes and the outbound IPsec Security Association Parameters on Initiator*

Note that the **ipv4 or ipv6** keyword, used with **tunnel mode** command, determines if the tunnel source and destination address is IPv4 or IPv6.

The following are the P2P tunnel interface commands on an initiator that determine the outbound IPsec Security Association negotiation parameters.

- The **tunnel mode** command determines the encapsulation protocol and the transport IP address family (IPv4 or IPv6). The addresses specified in **tunnel source** and **tunnel destination** commands must match this IP address family.
- The **tunnel source** address is typically the address of the WAN interface. When there are multiple WAN interfaces, it can be a loopback interface IP address that can be reached through all the WAN transports.
- The **ip address** and **ipv6 address** commands determine the overlay IP address family (IPv4 or IPv6) i.e whether the interface can support IPv4 and/or IPv6 overlay traffic.
- The **vrf forwarding** command specifies the IVRF (overlay VRF). The **tunnel vrf** command specifies the FVRF (transport VRF).

- The **tunnel protection** command enables IPsec on the interface and specifies the IPsec protection profile.

The following example illustrates the configuration of a P2P GRE tunnel interface with IPv4 transport, where the tunnel source and tunnel destination addresses are IPv4. Note that **tunnel mode gre ip** is not displayed in the running configuration as it is the default. The interface is enabled for both IPv4 and IPv6 overlay traffic as it is configured with both IPv4 and IPv6 addresses.

Click here to view code image

```
interface tunnel1
 vrf forwarding ivrf_1
 ip address 192.168.1.1 255.255.255.0
 ipv6 address 2001::1:1/120
 tunnel source Ethernet0/0
 tunnel destination 192.168.2.1
 tunnel vrf fvrf_1
 tunnel protection ipsec profile default
end

Router#show interfaces tunnel 1 | include Tunnel protocol/transport
  Tunnel protocol/transport GRE/IP
```

The other encapsulation modes and transport types can be configured on the tunnel interface as follows:

Click here to view code image

```
Router(config)#interface tunnel 1
Router(config-if)#tunnel mode gre ?
  ip          over IP
  ipv6        over IPv6

Router (config-if)#tunnel mode ipsec ?
  ipv4  over IPv4
  ipv6  over IPv6
```

Configuring Virtual-Template Interfaces

On a responder, the IPsec Security Association terminates on an IPsec interface, which in the case of FlexVPN is a tunnel interface with **tunnel protection** applied. The tunnel interface configuration and especially the tunnel encapsulation mode determine the incoming IPsec Security Associations proposals that an interface can accept. Table 8-3 shows the tunnel encapsulation modes and the incoming IPsec Security Association proposals that an IPsec interface can accept.

Tunnel Encapsulation Mode	Acceptable Inbound IPsec Security Association proposals
tunnel mode gre	IPsec peer addresses in the incoming proposal: Source peer address = tunnel destination address Destination peer address = tunnel source address IPsec Security Association traffic selectors in the incoming proposal: Local address = tunnel destination address Local mask = host mask Remote address = tunnel source address Remote mask = host mask Protocol = GRE (47) Local/Remote ports = Not applicable Note: This is referred to as a *GRE host-to-host SA*
tunnel mode IPsec	IPsec peer addresses in the incoming proposal: Source peer address = tunnel destination address Destination peer address = tunnel source address IPsec Security Association traffic selectors: Local address = any Local mask = any Remote address = any Remote mask = any Protocol = any Local/Remote ports = any Note: This is referred to as an *any-to-any SA*

Table 8-3 *Tunnel Encapsulation Modes and Acceptable IPsec Security Association Proposals on Responder*

Note that the **ipv4 or ipv6** keyword used with **tunnel mode** command determines if the tunnel source and destination addresses are IPv4 or IPv6.

On the responder a P2P tunnel interface is dynamically instantiated for every incoming IKEv2/IPsec session. This interface is known as a *virtual-access interface* that derives its configuration from the virtual-template interface of type tunnel configured in the IKEv2 profile associated with that incoming session. Note that the virtual-access interface can optionally derive its configuration from AAA that is obtained as part of user and/or group authorization described later in the chapter.

The following are the virtual-template configurations that determine the incoming IPsec Security Association proposals, that the cloned virtual-access interfaces can accept:

- The **tunnel mode** command specifies the encapsulation protocol and the transport IP address family (IPv4 or IPv6). The IPsec Security Association source and destination addresses in the incoming proposal must match this IP address family. The encapsulation mode determines the acceptable IPsec Security Association traffic selectors.

- The **tunnel destination** command must not be configured on a virtual-template. The cloned virtual-access interface derives its tunnel destination address from the source address of the incoming IPsec Security Association proposal.
- The **tunnel source** command may optionally be configured on a virtual-template, and when configured, it must match the destination address of the incoming IPsec Security Association proposal. Note that on a mismatch, the negotiation is aborted. When not configured, the cloned virtual-access interface derives the tunnel source address from the destination address of the incoming IPsec Security Association proposal.
- The **ip unnumbered** and **ipv6 unnumbered** commands determine the overlay IP address family (IPv4 or IPv6)—and therefore determine if the cloned virtual-access interface can support IPv4 and/or IPv6 overlay traffic.
- The **vrf forwarding** command specifies the IVRF (overlay VRF), and the **tunnel vrf** command specifies the FVRF (transport VRF).
- The **tunnel protection** command enables IPsec on the interface and specifies the IPsec protection profile.

Note that the IP address on a virtual-template must be configured, using **ip unnumbered *interface*** and/or **ipv6 unnumbered *interface*** commands. All the cloned virtual-access interfaces inherit these commands, which indicate that the IP address is borrowed from the specified interface and reused. This is required as Cisco IOS does not allow the same IP subnet to be configured, using **ip address** and **ipv6 address** commands to be used on more than one interface. In other words, if a virtual-template had the IP addresses configured using **ip address** and **ipv6 address** commands, the cloning of these commands on the instantiated virtual-access interfaces would fail with the error **"% *subnet* overlaps with Virtual-Template *n*."**

The following example shows the configuration of a virtual-template interface of type tunnel with encapsulation mode of P2P GRE and transport type IPv4. The **tunnel mode gre ip** is not displayed in the running configuration because it is the default. Note that **tunnel source** and **tunnel destination** commands are not configured. The interface is enabled for both IPv4 and IPv6 overlay traffic because it has configured both IPv4 and IPv6 addresses. The virtual-template must be referenced from the IKEv2 profile.

Click here to view code image

```
crypto ikev2 profile ikev2_profile
 virtual-template 1

interface virtual-template1 type tunnel
 ip unnumbered Ethernet0/0
 ipv6 unnumbered Ethernet0/0
 tunnel protection ipsec profile default

Router#show interfaces virtual-template 1 | include Tunnel protocol
  Tunnel protocol/transport GRE/IP
```

The other encapsulation modes and transport types can be configured on the virtual-template as follows:

Click here to view code image

```
Router(config)#interface virtual-template 1 type tunnel
Router(config-if)#tunnel mode gre ?
  ip          over IP
  ipv6        over IPv6

Router (config-if)#tunnel mode ipsec ?
  ipv4  over IPv4
  ipv6  over IPv6
```

Auto-Detection of Tunnel Encapsulation and Transport

FlexVPN supports automatic detection of tunnel encapsulation mode and transport IP protocol type on responders, such as the FlexVPN server, that enables them to support initiators using multiple encapsulation modes and transport IP types, using a single virtual-template interface. The auto-detect mode is configured in the IKEv2 profile that overrides the configuration obtained from the virtual-template. The cloned virtual-access interfaces instead derive the encapsulation modes and transport IP types from the incoming IPsec Security Association proposal. The following example shows tunnel encapsulation and transport auto-detection configuration.

Click here to view code image

```
Router(config)#crypto ikev2 profile ikev2_profile
Router(config-ikev2-profile)#virtual-template 1 ?
  mode  Enabling Tunnel Auto Mode
  <cr>

Router(config-ikev2-profile)#virtual-template 1 mode ?
  auto  Auto - Enable Auto feature on the ike profile

Router(config-ikev2-profile)#virtual-template 1 mode auto
Warning! Mode auto takes precedence over the mode configured on the
Virtual-Template
```

The following example shows cloning of virtual-access interface from a virtual-template, including the IKEv2 profile and virtual-template configuration.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share
 authentication remote pre-share
 keyring local ikev2_keyring
 virtual-template 1 mode auto

interface Virtual-Template1 type tunnel
 ip unnumbered Ethernet0/0
 tunnel protection ipsec profile default
```

Following is selected output from **debug crypto ikev2** and **debug vtemplate cloning** logs illustrating the cloning of virtual-access from virtual-template:

Click here to view code image

```
IKEv2:(SESSION ID = 17,SA ID = 1):Processing IKE_AUTH message
IKEv2:% DVTI create request sent for profile default with PSH index 1.
 %LINEPROTO-5-UPDOWN: Line protocol on Interface Virtual-Access1, changed state to
down
VT[Vi1]:Clone Vaccess from Virtual-Template1 (96 bytes)
```

```
VT[Vi1]:tunnel mode ipsec ipv6
VT[Vi1]:tunnel protection ipsec profile default
VT[Vi1]:end
IKEv2:% DVTI Vi1 created for profile default with PSH index 1.
```

The following **show crypto session** command output shows that the crypto session is associated with the virtual-access interface as highlighted:

Click here to view code image

```
Router#show crypto session
Crypto session current status
Interface: Virtual-Access1
Profile: default
Session status: UP-ACTIVE
Peer: 172.16.1.1 port 500
  Session ID: 15
  IKEv2 SA: local 172.16.2.1/500 remote 172.16.1.1/500 Active
  IPSEC FLOW: permit 47 host 172.16.2.1 host 172.16.1.1
        Active SAs: 2, origin: crypto-map
```

The following **show** command output illustrates the virtual-access configuration partly derived from virtual-template and partly derived from the incoming IPsec session. Note that the 'tunnel source' and 'tunnel destination' addresses are derived from the destination and source addresses in the incoming IPsec proposal.

Click here to view code image

```
Router#show derived-config interface virtual-access 1
Building configuration...
Derived configuration : 218 bytes

interface Virtual-Access1
 ip unnumbered Ethernet0/0
 tunnel source 172.16.2.1
 tunnel destination 172.16.1.1
 tunnel protection ipsec profile default
 no tunnel protection ipsec initiate
end
```

Benefits of Per-Peer P2P Tunnel Interfaces

As we have seen, FlexVPN uses a dedicated P2P tunnel interface for every session or peer. Most of the Cisco IOS features are interface-centric. That means the features are applied on an interface such as quality of service (QoS), access control List (ACL), and firewall. A key advantage of using a per-peer or per-session interface is that all the interface features become available on a per-peer basis without requiring any additional feature specific implementation that is required with multipoint interfaces. For example, a different QoS policy or ACL can be used for every peer by just configuring the QoS policy or ACL on the interface associated with that peer.

Because of per-peer P2P tunnel interface, FlexVPN allows multiple peers to be situated behind the same network address port translation (NAPT) device even when multiple peers have the same public address and/or same private address. This is currently not possible with DMVPN that uses multipoint interfaces.

Cisco IOS AAA Infrastructure

Cisco IOS AAA infrastructure provides an abstracted framework for authentication, authorization, and accounting using the local database or external AAA servers that hold the authentication credentials, authorization policy parameters and accounting data in a way that is transparent to the AAA clients. Please refer Cisco IOS AAA documentation available at http://www.cisco.com/ for further details. FlexVPN registers as an AAA client and leverages AAA for EAP authentication, AAA-based pre-shared keys, user and group authorization policy, and accounting. Table 8-4 lists the FlexVPN AAA operations and the AAA databases supported for each.

FlexVPN AAA Operation	Supported AAA Database
EAP authentication	External AAA server for standards-based EAP
	External and local AAA for Anyconnect-EAP
AAA-based pre-shared keys	External AAA server
User authorization	External AAA server and local AAA database[1]
Group authorization	External AAA server and local AAA database
Implicit authorization	External AAA server
Accounting	External AAA server

[1]User authorization with local AAA fixed in CSCtw74492.

Table 8-4 *FlexVPN AAA Operations and Supported AAA Databases*

The steps to configure AAA for FlexVPN are:

Step 1. Enable AAA, using the **aaa new-model** command.

Step 2. Define AAA method list that specifies an ordered list of AAA databases (local AAA database and external AAA servers) to be used for authentication, authorization, and accounting. Note that the next database in the ordered list is tried only if the previous database is not reachable and not if it returns a failure.

Step 3. If the method list specifies external AAA server groups, define the server groups and the servers.

Step 4. If the method list specifies local AAA database, configure the local AAA database for FlexVPN using the **crypto ikev2 authorization policy** command.

Step 5. Reference the AAA method list from the IKEv2 profile.

Configuring AAA for FlexVPN

The following example illustrates the configuration of AAA for FlexVPN. Note that the IKEv2 authorization policy configured, using **crypto ikev2 authorization policy** command, is the local AAA database for FlexVPN. The authorization policy supports all the policy attributes that can be defined on a remote authentication dial-in user service (RADIUS) server. The policy attributes can be defined through sub-commands and/or through an AAA attribute list.

Step 1. Enable AAA.

```
aaa new-model
```

Step 2. Define the AAA method lists for authentication, authorization, and accounting as needed.

Click here to view code image

```
aaa authentication login authen_list_radius group radius_group
aaa authorization network author_list_local local
aaa authorization network author_list_radius group radius_group
aaa accounting network acc_list_radius group radius_group
```

Step 3. Define the RADIUS server group.

Click here to view code image

```
aaa group server radius radius_group
 server name radius_server
```

Step 4. Define the RADIUS server.

Click here to view code image

```
radius server radius_server
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_key
```

Step 5. Configure local AAA database for FlexVPN.

Click here to view code image

```
crypto ikev2 authorization policy author_policy
 route set interface
 route accept any
 pool ip_address_pool
 aaa attribute list attr_list
```

Step 6. Define the AAA attribute list.

Click here to view code image

```
aaa attribute list attr-list1
attribute type interface-config "ip mtu 1100"
attribute type interface-config "tunnel key 10"
```

Step 7. Configure FlexVPN authentication, authorization, and accounting in the IKEv2 profile as needed.

Click here to view code image

```
crypto ikev2 profile ikev2_profile
 aaa authentication eap authen_list_radius
 aaa authorization user cert list author_list_radius user1
 aaa authorization group cert list author_list_local author_policy
 aaa accounting cert acc_list_radius
```

IKEv2 Name Mangler

The IKEv2 name mangler extracts a name string from the authenticated peer IKE identity string. The extracted name string is used as an AAA username with AAA authorization to perform policy lookup for the peer.

The name mangler offers the flexibility to perform AAA-based policy lookup for the peer based on arbitrary portions of the peer IKE identities of various types. The name mangler is referenced from

the IKEv2 profile specifically from the **aaa authorization** and **keyring aaa** commands that use AAA authorization for policy lookup.

Note that because AAA-based policy lookup must be based on an authenticated peer IKE identity, AAA authorization and hence the name mangler invocation is performed after peer authentication. When invoked, the name mangler takes an IKE identity string as input and outputs a name string based on the extraction rule configured in the name mangler for that identity type. Figure 8-1 illustrates the functionality provided by the IKE2 name mangler.

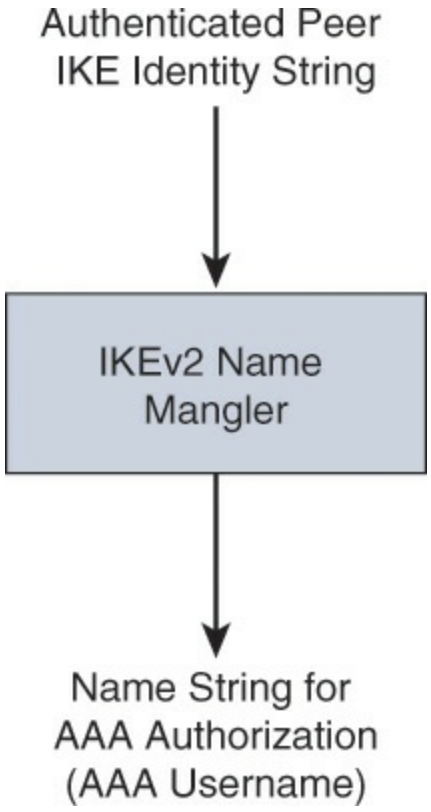


Figure 8-1 *IKEv2 Name Mangler*

Configuring IKEv2 Name Mangler

The IKEv2 name mangler is created using the following command in the router global configuration mode.

Click here to view code image

```
crypto ikev2 name-mangler name_mangler
```

Once the name mangler is created, the router command prompt will change from `(config)` to `(config-ikev2-name-mangler)`, denoting that user is in IKEv2 name mangler submode.

Table 8-5 shows the IKE identity types that can be configured in the IKEv2 name mangler.

IKE Identity Type	Description
FQDN	A fully qualified domain name string. An example of an FQDN is "example.com"
Email	A fully qualified RFC822 email address string. An example of a ID_RFC822_ADDR is "jsmith@example.com"
Distinguished Name (DN)	The binary distinguished encoding rules (DER) encoding of an ASN.1 X.500 distinguished name
EAP Identity	The EAP identity depends on the EAP method and can be of type FQDN, email, or DN

Table 8-5 *IKE Identity Types Supported by the IKEv2 Name Mangler*

The following example illustrates configuring an IKEv2 name mangler. The name mangler can have multiple identity statements, one of each type, and this allows the same name mangler to be used with multiple peers with different IKE identities.

Click here to view code image

```
Router(config-ikev2-name-mangler)#?
IKEv2 name mangler commands:
  dn      Derive name from DN identity
  eap     Derive name from EAP identity
  email   Derive name from EMAIL identity
  fqdn    Derive name from FQDN identity

Router#show run | begin name-mangler
crypto ikev2 name-mangler name_mangler
 fqdn hostname
 email domain
 dn common-name
 eap all
```

The name mangler must be referenced from the IKEv2 profile as part of the **aaa authorization** and **keyring aaa** commands. The same name mangler can be referenced from multiple commands in the IKEv2 profile.

Click here to view code image

```
crypto ikev2 profile ikev2_profile
 keyring aaa aaa_method_list name-mangler name_mangler
 aaa authorization group psk list aaa_list name-mangler name_mangler
 aaa authorization user psk list aaa_list name-mangler name_mangler
```

The name mangler must be dereferenced from the IKEv2 profile before it can be deleted:

Click here to view code image

```
Router(config)#no cryto ikev2 name-mangler name_mangler
% Cannot remove as name mangler is in use.
```

Extracting Name from FQDN Identity

The name mangler can be configured to extract the name string for AAA policy lookup from the hostname or domain portions of the peer FDQN identity string or the entire identity string. Table 8-6 lists the IKEv2 name mangler commands to extract the name from the FQDN identity.

IKEv2 Name Mangler Subcommand	Derived Name String
fqdn hostname	Extracts name string from the hostname portion of the FQDN string. For example, if the FQDN string is "user.domain" the name extracted will be "user."
fqdn domain	Extracts name string from the domain portion of the FQDN string. For example, if the FQDN string is "user.domain" the name extracted will be "domain." The portion after the first "." in the string is treated as domain.
fqdn all	Returns the entire FQDN string as the name string.

Table 8-6 *Extracting Name from IKEv2 FQDN Identity*

Extracting Name from Email Identity

The name mangler can be configured to extract the name string for AAA policy lookup from the username or domain portions of the peer email identity string or the entire identity string. Table 8-7 lists the IKEv2 name mangler commands to extract name from the email identity.

IKEv2 Name Mangler Subcommand	Derived Name String
email username	Extracts name string from the username portion of the email string. For example, if the email string is user@domain, the name extracted will be "user."
email domain	Extracts name string from the domain portion of the email string. For example, if the email string is "user@domain" the name extracted will be "domain". The portion after the first "@" in the string is treated as domain.
email all	Returns the entire email string as the name string.

Table 8-7 *Extracting Name from IKEv2 Email Identity*

Extracting Name from DN Identity

The name mangler can be configured to extract the name string for AAA policy lookup from a portion of the peer DN identity string or the entire identity string. Table 8-8 lists the IKEv2 name mangler commands to extract name from the DN identity.

IKEv2 Name Mangler Subcommand	Derived Name String
dn common-name	Extracts name string from the common-name (CN) portion of the DN string.
dn domain	Extracts name string from the domain portion of the DN string.
dn organization-unit	Extracts name string from the first organization-unit (OU) portion of the DN string.
dn organization	Extracts name string from the organization (O) portion of the DN string.
dn locality	Extracts name string from the locality (L) portion of the DN string.
dn state	Extracts name string from the state (S) portion of the DN string.
dn country	Extracts name string from the country (C) portion of the DN string.
dn all	Returns the entire DN string as the name string.

Table 8-8 *Extracting Name from IKEv2 DN Identity*

Extracting Name from EAP Identity

The name mangler can be configured to extract the name string for AAA policy lookup from a portion of the peer extensible authentication protocol (EAP) identity string or the entire identity string. Table 8-9 lists the IKEv2 name mangler commands to extract name from the EAP identity. The format of EAP identity depends on the EAP method.

IKEv2 Name Mangler Subcommand	Derived Name String
eap prefix delimiter . eap prefix delimiter @ eap prefix delimiter \	Extracts name string from the prefix portion of the EAP identity string, based on the specified delimiter.
eap prefix delimiter . eap prefix delimiter @ eap prefix delimiter \	Extracts name string from the suffix portion of the EAP identity string, based on the specified delimiter.
eap all	Returns the entire EAP identity string as the name string.
eap dn common-name eap dn domain eap dn common-name eap dn organization-unit eap dn organization eap dn locality eap dn state eap dn country	Extracts name string from the specified portion of the EAP identity string when it is of type DN.

Table 8-9 *Extracting Name from IKEv2 DN Identity*

IKEv2 Authorization Policy

The IKEv2 authorization policy is FlexVPN's local AAA database that is used with FlexVPN authorization. The authorization policy supports all the policy attributes that can be defined on an external AAA server. The list of all the FlexVPN AAA attributes and their description is covered in Chapter 9 "FlexVPN Server." The following example illustrates configuring an IKEv2 authorization policy along with its use with FlexVPN authorization.

The IKEv2 authorization policy is defined using the **crypto ikev2 authorization policy** command in the router global configuration mode. The policy attributes are defined using subcommands and/or through a AAA attribute list.

Click here to view code image

```
Router(config)#crypto ikev2 authorization policy ikev2_author_policy
Router(config-ikev2-author-policy)#?
IKEv2 authorization policy commands:
  aaa                           Specify aaa attribute list
  backup-gateway                Specify backup gateway
  banner                        Specify mode config banner
  configuration                 Push configuration to the client
  def-domain                    Set default domain name to send to client
  dhcp                          Specify DHCP server for config address
                                assignment
  dns                           Specify DNS Addresses
  exit                          Exit from crypto ikev2 author policy sub mode
  include-local-lan             Enable Local LAN Access with no split tunnel
  ipsec                         Specify ipsec parameters
  ipv6                          Specify the ipv6 attributes
  mobileopt-serv                Specify Mobile Optimization Server Addr and
```

```
                                Port
  netmask                       Specify netmask of the config address
  no                            Negate a command or set its defaults
  pfs                           The client should propose PFS
  pool                          Specify local address pool
  route                         Specify route parameters
  session-lifetime              Specify maximum session lifetime
  smartcard-removal-disconnect  Enables smartcard-removal-disconnect
  split-dns                     Split DNS names
  wins                          Specify WINS Addresses
```

The policy attributes can also be defined, using an AAA attribute list that can be referenced from the IKEv2 authorization policy.

Click here to view code image

```
Router(config-ikev2-author-policy)#aaa attribute list ?
  WORD  AAA attribute list name
```

The AAA attribute list supports all the RADIUS attributes, however they must be configured in the Cisco IOS AAA format. Use the **show aaa attribute protocol radius** command to get the Cisco IOS AAA format of the RADIUS attributes. Please refer Cisco IOS documentation for "AAA attribute list" for further details.

Click here to view code image

```
aaa attribute list attr-list1
 attribute type interface-config "ip mtu 1100"
 attribute type interface-config "tunnel key 10"
```

The following example illustrates the AAA commands required along with the IKEv2 authorization policy, and referencing the policy from IKEv2 profile.

Configure AAA method lists for authorization, using local AAA database.

Click here to view code image

```
aaa new-model
aaa authorization network aaa_list1 local
```

Configure FlexVPN authorization in the IKEv2 profile as highlighted in the following example.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote any
 identity local dn
 authentication local pre-share
 authentication remote pre-share
 aaa authorization group psk list aaa_list1 ikev2_author_policy
 aaa authorization user psk list aaa_list1 ikev2_author_policy
```

Note that, during FlexVPN authorization based on local AAA database, the IKEv2 authorization policies are looked up based on the name string specified in the **aaa authorization group** command or based on the name string extracted from the peer IKE identity by the name mangler specified in the command.

The configured and the default IKEv2 authorization policies can be displayed, using the **show crypto ikev2 authorization policy** command.

Click here to view code image

```
Router#show crypto ikev2 authorization policy
 IKEv2 Authorization Policy : default
  route set interface
  route accept any tag : 1 distance : 1
 IKEv2 Authorization Policy : ikev2_author_policy
  IPV4 Address Pool : ip-addr-pool
  IPv4 DNS Primary : 192.168.10.10
  route accept any tag : 1 distance : 1
```

Default IKEv2 Authorization Policy

The default IKEv2 authorization policy consists of commonly used attributes. The default policy must be referenced from the **aaa authorization** commands in the IKEv2 profile. The default policy can be disabled or modified to address the specific needs of users after which it is displayed in the running configuration. The default authorization policy is used by most customers without amendments, The following example shows disabling, modifying, and restoring of default IKEv2 authorization policy.

The default IKEv2 authorization policy can be displayed, using the following commands.

Click here to view code image

```
Router#show run all | section ikev2 authorization policy
crypto ikev2 authorization policy default
 route set interface
 route accept any

Router#show crypto ikev2 authorization policy default
 IKEv2 Authorization Policy : default
  route set interface
  route accept any tag : 1 distance : 1
```

The **route set interface** command will advertise the IP address through the configuration payload that is obtained from the tunnel or virtual-access interface or from the specified interface. The **route accept any** command will install any prefixes sent from the peer, using configuration payload.

The default IKEv2 authorization policy is disabled, using the following command:

Click here to view code image

```
Router(config)#no crypto ikev2 authorization policy default

Router#show run | include ikev2 authorization policy default
no crypto ikev2 authorization policy default

Router#show crypto ikev2 authorization policy default
 IKEv2 Authorization Policy : default Disabled
```

The default authorization policy is re-enabled, using the following command:

Click here to view code image

```
Router(config)#default crypto ikev2 authorization policy
```

The default authorization policy is modified as shown in the following code, after which it is displayed in running configuration.

Click here to view code image

```
Router(config)#crypto ikev2 authorization policy default
%Warning: This will Modify Default IKEv2 Authorization Policy. Exit if you don't
want
Router(config-ikev2-author-policy)#pool ip-address-pool
Router(config-ikev2-author-policy)#dns 192.168.6.10

Router#show run | begin ikev2 authorization policy default
crypto ikev2 authorization policy default
 pool ip-address-pool
 dns 192.168.6.10

Router#show crypto ikev2 authorization policy default
 IKEv2 Authorization Policy : default
  IPV4 Address Pool : ip-address-pool
  IPv4 DNS Primary : 192.168.6.10
  route set interface
  route accept any tag : 1 distance : 1
```

The modified default authorization policy can be restored as follows:

Click here to view code image

```
Router(config)#default crypto ikev2 authorization policy
```

FlexVPN Authorization

FlexVPN authorization provides policy attributes for an IKEv2/IPsec session based on the authenticated peer IKE identity. FlexVPN uses either the local AAA database or an external AAA server to store the policy and uses AAA authorization for policy lookup. The policy obtained using AAA authorization consists of local and remote attributes. The remote attributes are pushed to the peer, using IKEv2 configuration exchange, and local attributes are applied locally to the session/interface corresponding to that peer. Implicit authorization occurs when attributes received during authentication are re-used for authorization.

Figure 8-2 illustrates FlexVPN authorization and highlights the key points that the authorization can be performed both on the FlexVPN initiator and responder and only after successful peer authentication. The AAA username used for authorization lookup is typically based on the authenticated peer IKE identity. The AAA infrastructure provides an abstracted interface to AAA database to the clients regardless of whether the database is local or external.

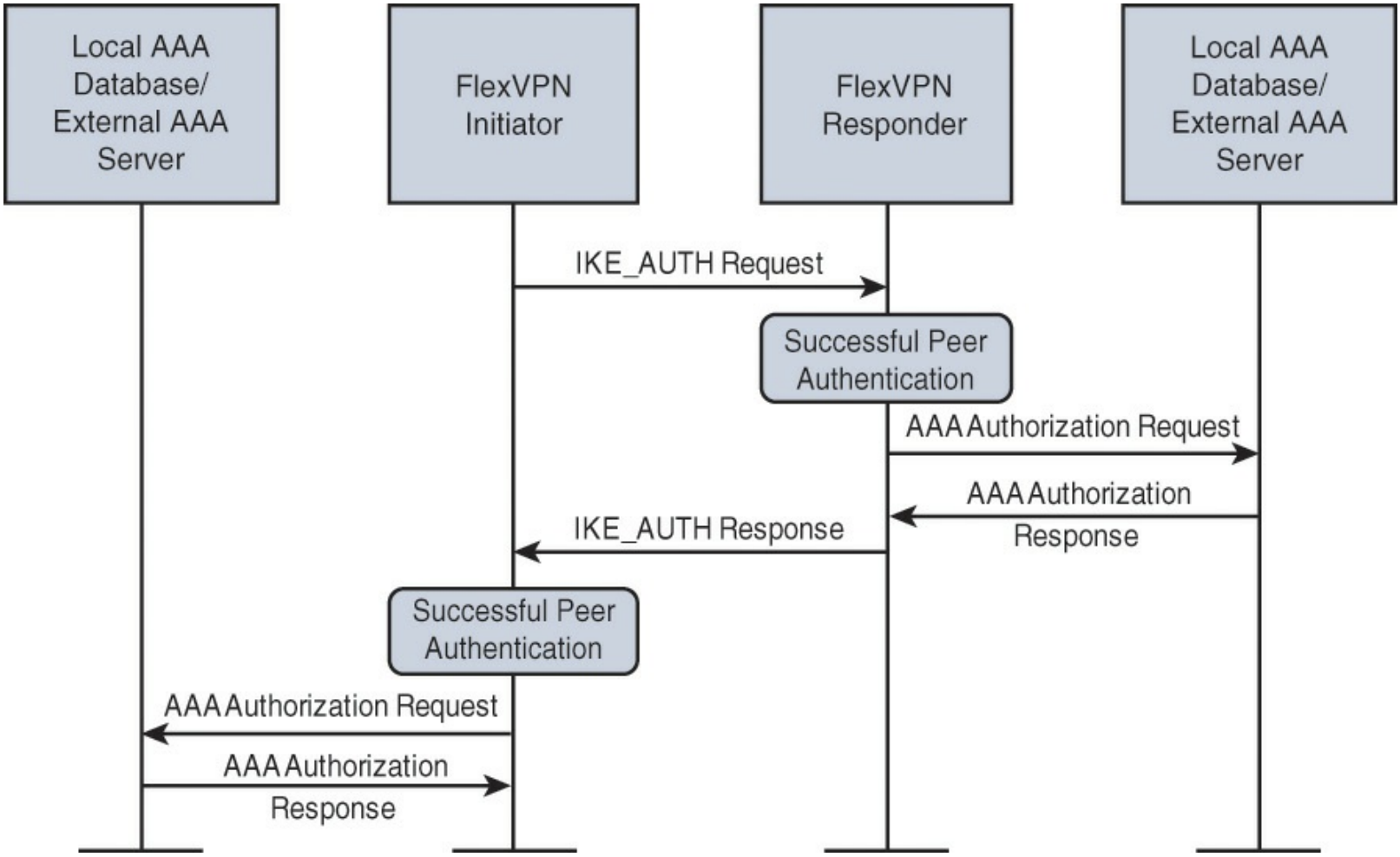


Figure 8-2 *FlexVPN Authorization*

FlexVPN supports three types of authorization. Table 8-10 lists the FlexVPN authorization types and the AAA databases they support.

FlexVPN Authorization Type	Supported AAA Database
User authorization	External AAA server and local AAA database[1]
Group authorization	External AAA server and local AAA database
Implicit authorization	External AAA server

[1]User authorization with local AAA fixed in 15.6T (CSCtw74492).

Table 8-10 *FlexVPN Authorization Types and Supported AAA Databases*

Figure 8-3 illustrates FlexVPN authorization, using a local AAA database. The IKEv2 authorization policy acts as the local AAA database for FlexVPN. IKEv2 invokes AAA with an authorization request specifying the AAA method list and the username. When the method list specifies local authorization, AAA invokes IKEv2 that looks up an IKEv2 authorization policy matching the username. After that, it returns the parameters configured under that authorization policy as AAA attributes, and then AAA returns these attributes back to IKEv2 in the authorization response.

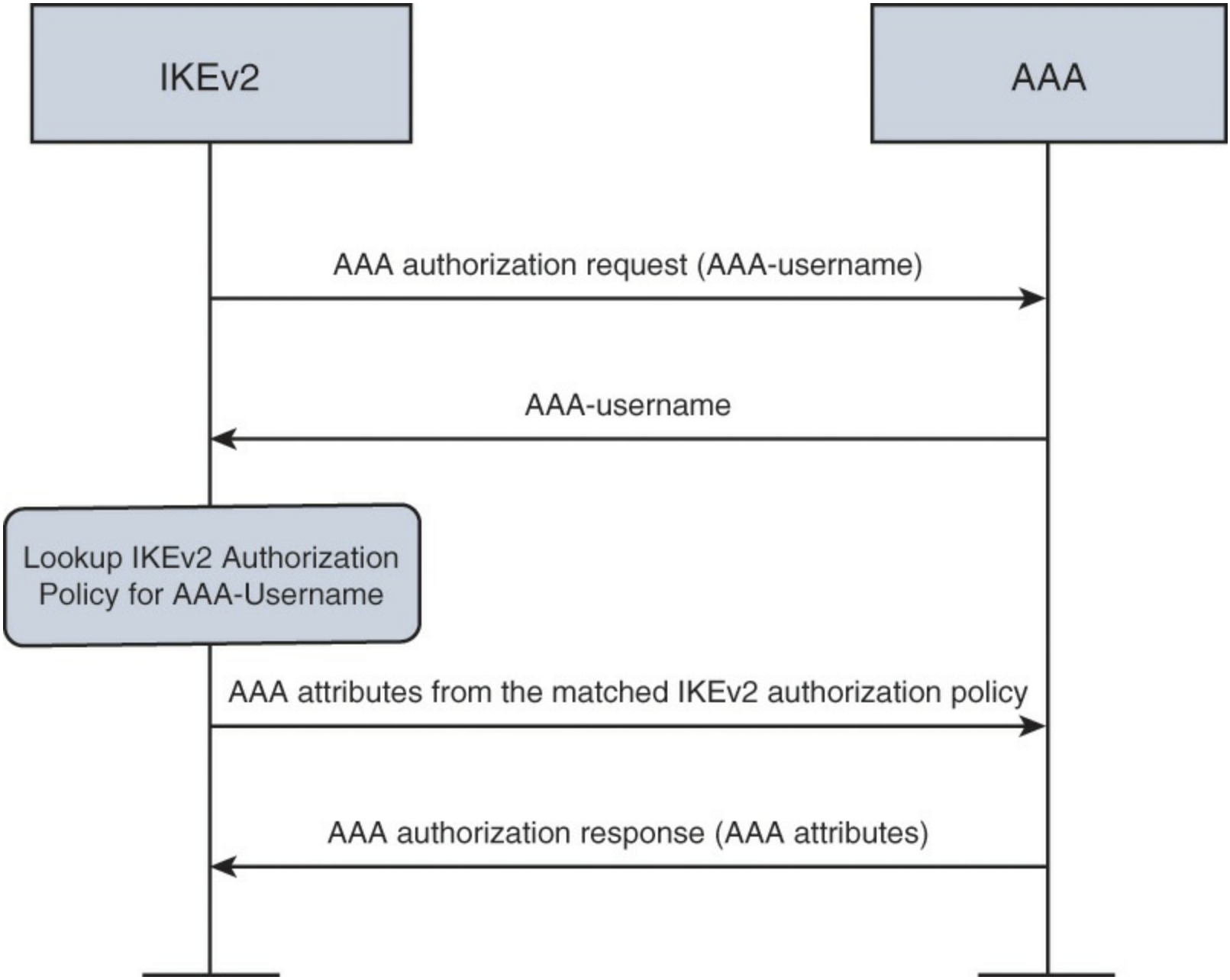


Figure 8-3 *FlexVPN Authorization using a Local AAA Database*

Figure 8-4 illustrates FlexVPN authorization, using a RADIUS server as an external AAA server. IKEv2 invokes AAA with an authorization request that specifies the AAA method list and the username. When the method list specifies RADIUS-based authorization, AAA acting as a RADIUS client sends an Access-Request to the RADIUS server. The RADIUS server looks up a policy matching the specified username and returns the corresponding RADIUS attributes. AAA converts the RADIUS attributes into AAA attributes and returns them to IKEv2 in authorization response.

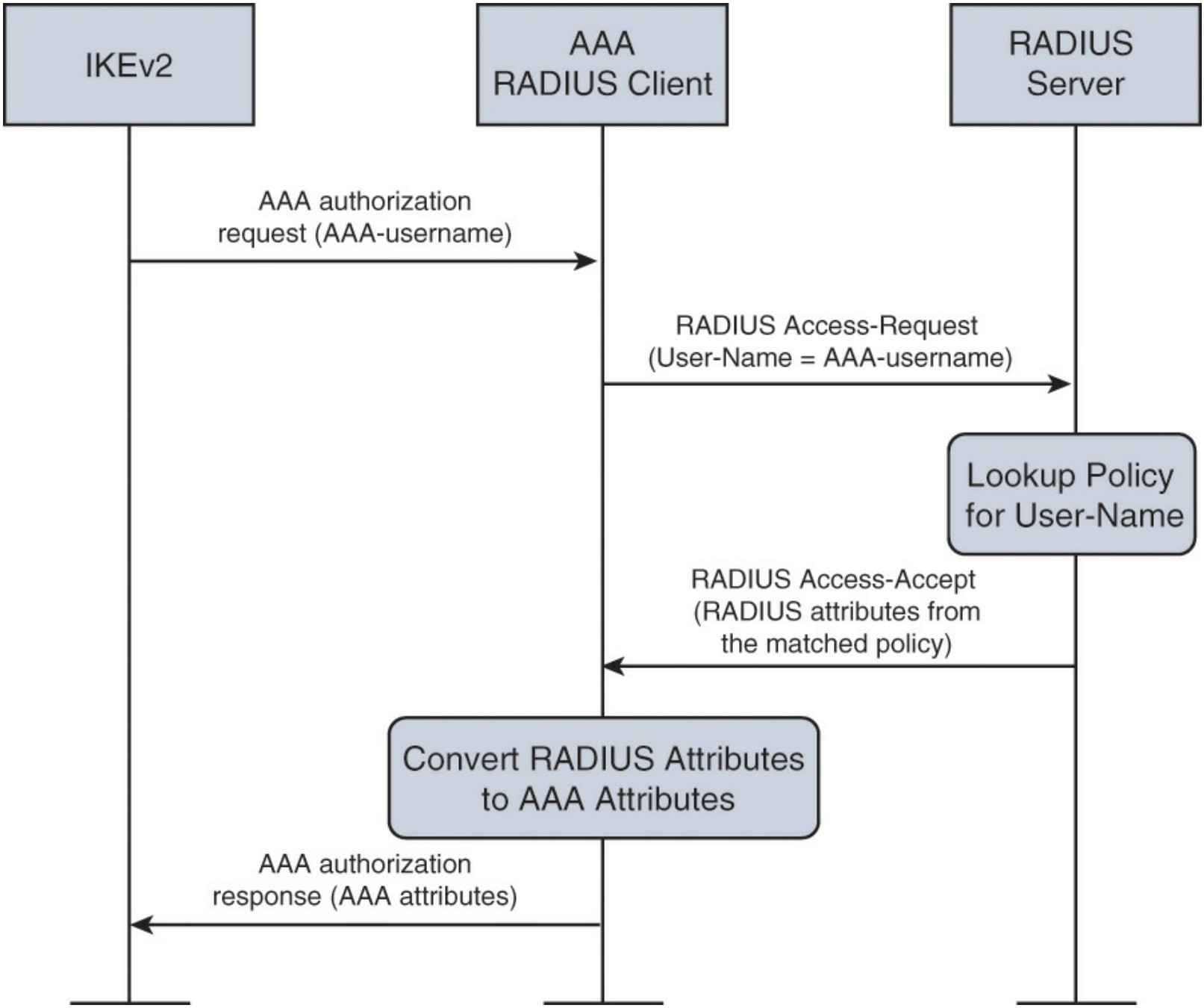


Figure 8-4 *FlexVPN authorization, using the RADIUS server*

Configuring FlexVPN Authorization

FlexVPN authorization is configured in the IKEv2 profile, using the **aaa authorization** command that specifies the AAA method list and the name string to be used as the AAA username for the authorization lookup. The AAA method list specifies the AAA databases that the authorization request will look up. The authorization name string can be directly specified or can be extracted from the peer IKE identity, using an IKEv2 name mangler. Each of the three FlexVPN authorization types can be configured multiple times, one for every peer authentication method. This allows the flexibility to use a different AAA database and name string, based on the peer authentication method. The following example illustrates configuring FlexVPN authorization.

Specify the authorization type:

Click here to view code image

```
Router(config-ikev2-profile)#aaa authorization ?
  group  AAA group authorization
```

```
  user   AAA user authorization
```

Specify the peer authentication method for which this configuration would be used:

Click here to view code image

```
Router(config-ikev2-profile)#aaa authorization user ?
  anyconnect-eap  AAA list to use when IKEv2 remote auth method is anyconnect
                  eap based
  cert            AAA list to use when IKEv2 remote auth method is certificate
                  based
  eap             AAA list to use when IKEv2 remote auth method is EAP
  override        Override user authorization with group authorization. By
                  default, group authorization is overridden with user
                  authorization
  psk             AAA list to use when IKEv2 remote auth method is PSK
```

The **cached** keyword enables implicit authorization. Note that with implicit authorization, the AAA method list and the name string for lookup are not specified. Implicit authorization can be useful when retrieving AAA keyrings containing authorization attributes, or when EAP is used. A major benefit of implicit authorization is that a single lookup is required, compared to a separate lookup once for authentication and once for authorization.

Click here to view code image

```
Router(config-ikev2-profile)#aaa authorization user psk ?
  cached  Use cached attributes from EAP authentication or AAA pre-shared key
          fetch
  list    AAA method list

Router(config-ikev2-profile)#aaa authorization user psk cached ?
  <cr>
```

Specify the AAA method list:

Click here to view code image

```
Router(config-ikev2-profile)#aaa authorization user psk list ?
  WORD  AAA list name
```

Specify the name string to be used for authorization lookup or specify a name mangler that will extract the name from peer IKE identity:

Click here to view code image

```
Router(config-ikev2-profile)#aaa authorization user psk list aaa_list ?
  WORD          AAA username
  name-mangler  Specify the name-mangler to derive AAA username
  password      Specify the AAA password
  <cr>

Router(config-ikev2-profile)#aaa authorization user psk list aaa_list
name-mangler ?
  WORD  mangler name

Router(config-ikev2-profile)#aaa authorization user psk list aaa_list name-mangler
name_mangler ?
  password  Specify the AAA password
  <cr>
```

Specify the password to authenticate to the AAA server:

Click here to view code image

```
Router(config-ikev2-profile)# aaa authorization user psk list aaa_list
name-mangler name_mangler password ?
  0     Specifies an UNENCRYPTED password will follow
  6     Specifies an ENCRYPTED password will follow
  WORD  The UNENCRYPTED (cleartext) user password
```

FlexVPN User Authorization

FlexVPN user authorization is used to provide session policy attributes that are specific to a user. The user authorization lookup is typically based on the portion of the authenticated IKE identity that is unique to a user, such as the hostname portion of the FQDN identity. The IKEv2 name mangler is used to extract the desired portion of IKE identity for policy lookup. FlexVPN user authorization is configured in the IKEv2 profile, using the **aaa authorization user** command.

FlexVPN user authorization is supported with both an external AAA server and a local AAA as the policy database. However, when user authorization policy needs to be configured for a large number of users, using an external AAA server is a more scalable option.

FlexVPN User Authorization, Using an External AAA Server

The following example illustrates FlexVPN user authorization configuration, using an external AAA server. The user authorization is configured for pre-shared key and certificate-based remote authentication methods, that is, for peer authentication, using pre-shared key or certificate-based methods. Note that the two user authorization statements use a different AAA method list, pointing to a different RADIUS server and a different name mangler to obtain the name string for authorization lookup. The authorization is performed, using the user authorization statement corresponding to the peer authentication method.

Step 1. Enable AAA.

```
aaa new-model
```

Step 2. Define RADIUS server groups.

Click here to view code image

```
aaa group server radius radius_group1
 server name radius_server1

aaa group server radius radius_group2
 server name radius_server2
```

Step 3. Define AAA method lists for authorization, using an external AAA server.

Click here to view code image

```
aaa authorization network aaa_list1 group radius_group1
aaa authorization network aaa_list2 group radius_group2
```

Step 4. Define the RADIUS servers.

Click here to view code image

```
radius server radius_server1
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_server1_key
```

```
radius server radius_server2
 address ipv4 172.16.1.4 auth-port 1645 acct-port 1646
 key radius_server2_key
```

Step 5. Define IKEv2 name manglers.

Click here to view code image

```
crypto ikev2 name-mangler name_mangler1
 fqdn hostname

crypto ikev2 name-mangler name_mangler2
 dn common-name
```

Step 6. Configure FlexVPN user authorization in the IKEv2 profile as highlighted in the following example:

Click here to view code image

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 authentication remote rsa-sig
 aaa authorization user psk list aaa_list1 name-mangler name_mangler1
 aaa authorization user cert list aaa_list2 name-mangler name_mangler2
```

The following example illustrates the FlexVPN user authorization process during IKEv2 negotiation, using the selected output from **debug crypto ikev2**, **debug aaa subsys**, and **debug radius authentication** logs. The user authorization is based on the configuration shown in the previous example. The relevant debug outputs are highlighted.

Note that the peer is authenticating by using the pre-shared-key method and is using an IKE identity of type FQDN with value **router1.domain1**.

Click here to view code image

```
IKEv2:(SESSION ID = 96,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 338E72677B66C73E - Responder SPI : 4C220FBE34587D45 Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST
IKEv2:(SESSION ID = 96,SA ID = 1):Peer's authentication method is 'PSK'
IKEv2:(SESSION ID = 96,SA ID = 1):Searching policy based on peer's identity
'router1.domain1' of type 'FQDN'
IKEv2:(SESSION ID = 96,SA ID = 1):Verification of peer's authenctication data
PASSED
```

Note that the authorization is performed after successful peer authentication. The user authorization command corresponding to the pre-shared-key method, **aaa authorization user psk**, is used. The server group ***radius_group1*** and the radius server ***172.16.1.3*** correspond to the specified AAA method list ***aaa_list1***. The name string "router1" used in the authorization request and the RADIUS Access-Request is extracted from the peer FQDN identity "router1.domain" by the specified name mangler ***name_mangler1***.

Click here to view code image

```
IKEv2:Using mlist aaa_list1 and username router1 for user author request
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorization request sent
```

```
AAA SRV(00000062): Author method=SERVER_GROUP radius_group1
RADIUS/ENCODE(00000062):Orig. component type = VPN IPSEC

RADIUS/ENCODE: Best Local IP-Address 172.16.1.2 for Radius-Server 172.16.1.3
RADIUS(00000062): Send Access-Request to 172.16.1.3:1645 id 1645/85, len 132
RADIUS:  User-Name           [1]   9   "router1"
RADIUS:  User-Password       [2]   18  *
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Service-Type        [6]   6   Outbound                  [5]
RADIUS:  NAS-IP-Address      [4]   6   172.16.1.2
RADIUS(00000062): Sending a IPv4 Radius Packet
RADIUS: Received from id 1645/85 172.16.1.3:1645, Access-Accept, len 128
RADIUS:   Cisco AVpair       [1]   27  "ipsec:route-set=interface"
RADIUS:   Cisco AVpair       [1]   24  "ipsec:route-accept=any"
RADIUS:   Cisco AVpair       [1]   39  "ipsec:route-set=prefix 192.168.1.1/24"
AAA SRV(00000062): protocol reply PASS for Authorization
AAA SRV(00000062): Return Authorization status=PASS
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorization response
```

FlexVPN Group Authorization

FlexVPN group authorization is used to provide session policy attributes that are common to a related set of peers. The group authorization lookup is typically based on the common portion of the authenticated IKE identity, such as the domain portion of the FQDN identity. The IKEv2 name mangler is used to extract the desired portion of the IKE identity for policy lookup. FlexVPN group authorization is configured in IKEv2 profile, using the **aaa authorization group** command.

FlexVPN group authorization is supported with both an external AAA server and a local AAA as policy database. A local AAA database is supported because the group authorization policy is common for a set of peers and hence is not likely to pose a scale problem.

FlexVPN Group Authorization, Using a Local AAA Database

The following example illustrates FlexVPN group authorization configuration, using a local AAA database. The group authorization is configured for a certificate-based remote authentication method, that is, for peer authentication, using the certificate-based method. The authorization command references an AAA method list that specifies the local AAA database. The referenced name mangler extracts the name string from the domain portion of the DN identity received from the peer.

Step 1. Enable AAA.

```
aaa new-model
```

Step 2. Define AAA method lists for authorization, using a local AAA database.

Click here to view code image

```
aaa authorization network aaa_list1 local
```

Step 3. Define IKEv2 name mangler.

Click here to view code image

```
crypto ikev2 name-mangler name_mangler1
 dn domain
```

Step 4. Define an IKEv2 authorization policy which acts as a local AAA database for FlexVPN.

Click here to view code image

```
crypto ikev2 authorization policy domain1
 route set interface
 route set remote ipv4 192.168.2.0 255.255.255.0
```

Step 5. Configure the FlexVPN group authorization in the IKEv2 profile as highlighted in the following example:

Click here to view code image

```
crypto ikev2 profile default
 match identity remote any
 identity local dn
 authentication local rsa-sig
 authentication remote pre-share
 authentication remote rsa-sig
 pki trustpoint ca_server1
 aaa authorization group cert list aaa_list1 name-mangler name_mangler1
```

The following example illustrates the FlexVPN group authorization process during the IKEv2 negotiation using the selected output from **debug crypto ikev2**, **debug crypto ikev2 internal**, and **debug aaa subsys** logs. The group authorization is based on the configuration shown in the previous example. The relevant debug outputs are highlighted.

Note that the peer is authenticating, using a certificate-based method and is using an IKE identity of type DN.

Click here to view code image

```
IKEv2:(SESSION ID = 42,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 91BDE56BC7C68593 - Responder SPI : 0BC460361D49F928 Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST

IKEv2:(SESSION ID = 42,SA ID = 1):Searching policy based on peer's identity 'hostn
ame=router1,cn=router1,dc=domain1' of type 'DER ASN1 DN'

IKEv2:(SESSION ID = 42,SA ID = 1):Peer's authentication method is 'RSA'

IKEv2:(SA ID = 1):[Crypto Engine -> IKEv2] Verification of signed authentication
data PASSED
```

Note that the authorization is performed after successful peer authentication. The group authorization command corresponding to the certificate method, **aaa authorization user cert**, is used. The name string *domain1* used in the authorization request and to look up the IKEv2 authorization policy is extracted from the peer DN identity by the specified name mangler *name_mangler1*.

Click here to view code image

```
IKEv2:Using mlist aaa_list1 and username domain1 for group author request
AAA/AUTHOR (0x21): Pick method list 'aaa_list1'
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorization request sent
AAA SRV(00000021): process author req
AAA SRV(00000021): Author method=LOCAL
AAA/LOCAL/AUTHEN: starting
AAA/LOCAL/AUTHEN(21): authorizing crypto domain1
IKEv2-INTERNAL:IKEv2 local AAA author request for 'domain1'
AAA SRV(00000021): protocol reply PASS for Authorization
AAA SRV(00000021): Return Authorization status=PASS
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorization response
```

```
IKEv2-INTERNAL:Received group author attributes:
route-set-interface: 1, route-set-remote-ipv4: 192.168.2.0 255.255.255.0
route-accept any tag:1 distance:1,
```

FlexVPN Group Authorization, Using an External AAA Server

The following example illustrates a FlexVPN group authorization configuration, using an external AAA server. The group authorization is configured for certificate-based remote authentication methods, that is, for peer authentication, using certificate-based methods. The group authorization statement references an AAA method list that specifies a RADIUS server group. The name mangler derives the name string for authorization lookup.

Step 1. Enable AAA.

```
aaa new-model
```

Step 2. Define RADIUS server groups.

Click here to view code image

```
aaa group server radius radius_group1
 server name radius_server1
```

Step 3. Define AAA method lists for authorization, using external AAA server.

Click here to view code image

```
aaa authorization network aaa_list1 group radius_group1
```

Step 4. Define the RADIUS servers.

Click here to view code image

```
radius server radius_server1
 address ipv4 172.16.1.4 auth-port 1645 acct-port 1646
 key radius_server1_key
```

Step 5. Define IKEv2 name mangler.

Click here to view code image

```
crypto ikev2 name-mangler name_mangler1
 dn domain
```

Step 6. Configure the FlexVPN group authorization in the IKEv2 profile as highlighted in the following example:

Click here to view code image

```
crypto ikev2 profile default
 match identity remote any
 identity local dn
 authentication local rsa-sig
 authentication remote pre-share
 authentication remote rsa-sig
 pki trustpoint ca_server1
 aaa authorization group cert list aaa_list1 name-mangler name_mangler1
```

The following example illustrates FlexVPN group authorization process during the IKEv2 negotiation, using the selected output from **debug crypto ikev2**, **debug aaa subsys**, and **debug radius**

authentication logs. The group authorization is based on the configuration shown in the previous example. The relevant debug outputs are highlighted.

Note that the peer is authenticating, using a certificate-based method and is using an IKE identity of type DN.

Click here to view code image

```
IKEv2:(SESSION ID = 61,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 5FC7B323505CC9F8 - Responder SPI : C8F23357295216E3 Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST

IKEv2:(SESSION ID = 61,SA ID = 1):Searching policy based on peer's identity 'hostn
ame=router1,cn=router1,dc=domain1' of type 'DER ASN1 DN'

IKEv2:(SESSION ID = 61,SA ID = 1):Peer's authentication method is 'RSA'

IKEv2:(SA ID = 1):[Crypto Engine -> IKEv2] Verification of signed authentication
data PASSED
```

Note that the authorization is performed after successful peer authentication. The server group ***radius_group1*** and the radius server ***172.16.1.4*** correspond to the specified AAA method list ***aaa_list1***. The name string **"domain"** used in the authorization request and the RADIUS Access-Request is extracted from the peer DN identity by the specified name mangler ***name_mangler1***.

Click here to view code image

```
IKEv2:Using mlist aaa_list1 and username domain1 for group author request
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorization request sent

AAA SRV(00000034): process author req
AAA SRV(00000034): Author method=SERVER_GROUP radius_group1
RADIUS/ENCODE(00000034):Orig. component type = VPN IPSEC
RADIUS/ENCODE: Best Local IP-Address 172.16.1.2 for Radius-Server 172.16.1.4
RADIUS(00000034): Send Access-Request to 172.16.1.4:1645 id 1645/19, len 132
RADIUS:  authenticator BA D7 AE 45 D6 33 76 21 - 20 F8 4F 7A 4C DE E5 1D
RADIUS:  User-Name           [1]   9   "domain1"
RADIUS:  User-Password       [2]   18  *
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  61
RADIUS:   Cisco AVpair       [1]   55  "audit-session-id=L2L4AC1G102ZO2L4AC1G101Z
I1F401F4ZN3D"
RADIUS:  Service-Type        [6]   6   Outbound                  [5]
RADIUS:  NAS-IP-Address      [4]   6   172.16.1.2
RADIUS(00000034): Sending a IPv4 Radius Packet
RADIUS(00000034): Started 5 sec timeout
RADIUS: Received from id 1645/19 172.16.1.4:1645, Access-Accept, len 128
RADIUS:  authenticator EF C6 8B 50 26 AE C2 F3 - 82 2A F1 07 B0 23 C2 ED
RADIUS:  Vendor, Cisco       [26]  33
RADIUS:   Cisco AVpair       [1]   27  "ipsec:route-set=interface"
RADIUS:  Vendor, Cisco       [26]  30
RADIUS:   Cisco AVpair       [1]   24  "ipsec:route-accept=any"
RADIUS:  Vendor, Cisco       [26]  45
RADIUS:   Cisco AVpair       [1]   39  "ipsec:route-set=prefix 192.168.1.1/24"
AAA SRV(00000034): protocol reply PASS for Authorization
AAA SRV(00000034): Return Authorization status=PASS
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response
IKEv2-INTERNAL:Received group author attributes:
route-set-interface: 1, route-accept any tag:1 distance:1,
```

FlexVPN Implicit Authorization

When FlexVPN uses an external AAA server, such as a RADIUS server for EAP authentication or to fetch AAA-based pre-shared keys, the AAA server may return all the configured policy attributes corresponding to the username used for EAP authentication or pre-shared key lookup. These unsolicited policy attributes are cached and used as user authorization policy attributes when the **aaa authorization user** command with '**cached**' option is configured. This is known as *implicit authorization*. When the '**cached**' option is not configured, these policy parameters are ignored. The implicit authorization avoids the need for an explicit user authorization and saves a round trip to the external AAA server.

Figure 8-5 illustrates FlexVPN implicit authorization with EAP authentication and a RADIUS-based EAP server. When EAP authentication is successful, the RADIUS server along with the EAP-Success message returns the policy attributes corresponding to the authenticated EAP username. When caching is enabled, these policy attributes are cached and used as implicit authorization attributes.

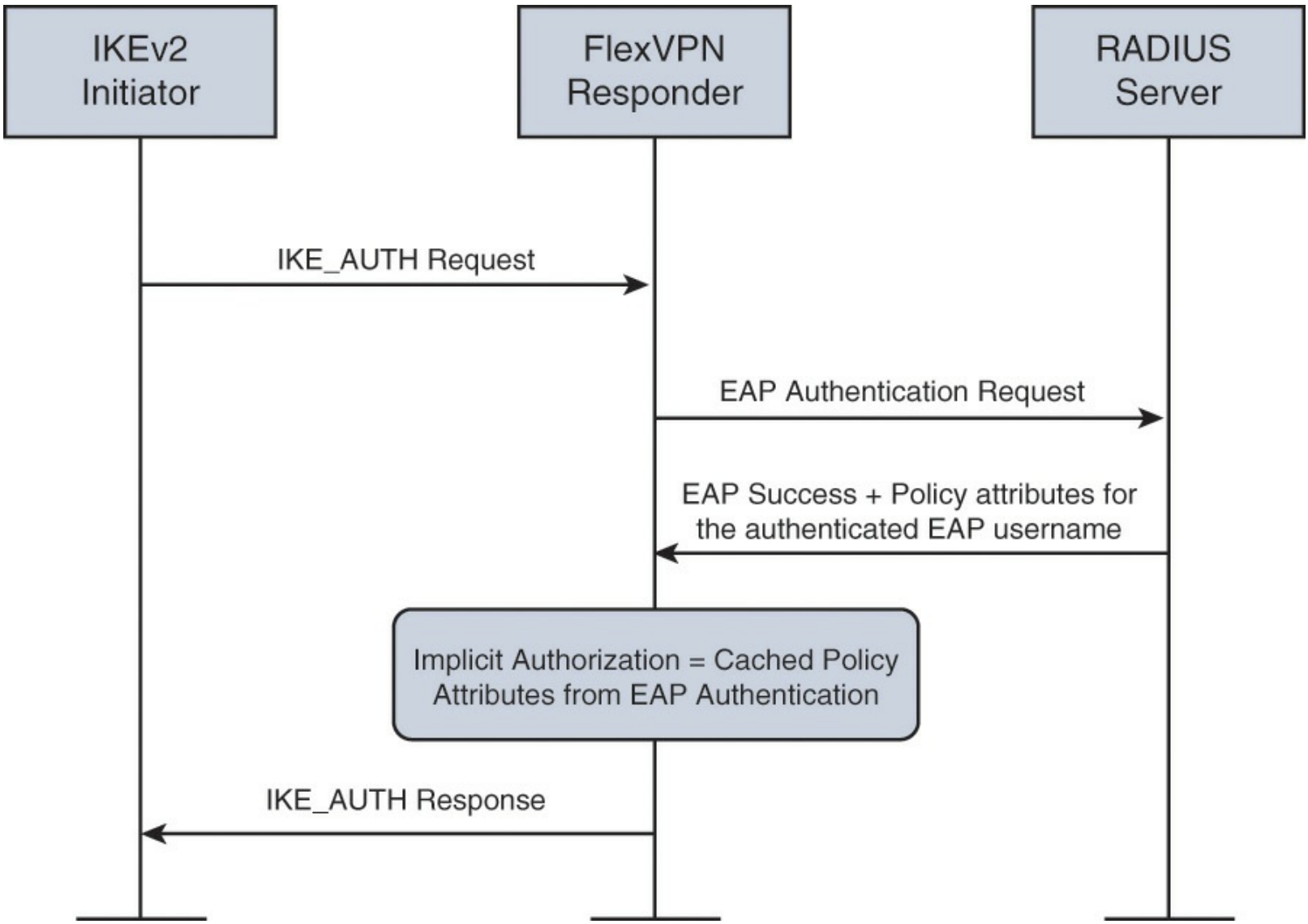


Figure 8-5 *FlexVPN Implicit Authorization with EAP Authentication*

Figure 8-6 illustrates FlexVPN implicit authorization with AAA-based, pre-shared keys hosted on a RADIUS server. When an AAA authorization request is sent to fetch the pre-shared keys for a username, the RADIUS server, along with the pre-shared keys, returns the policy attributes corresponding to the username used for the key lookup. When enabled, these policy attributes are cached and used as implicit authorization attributes.

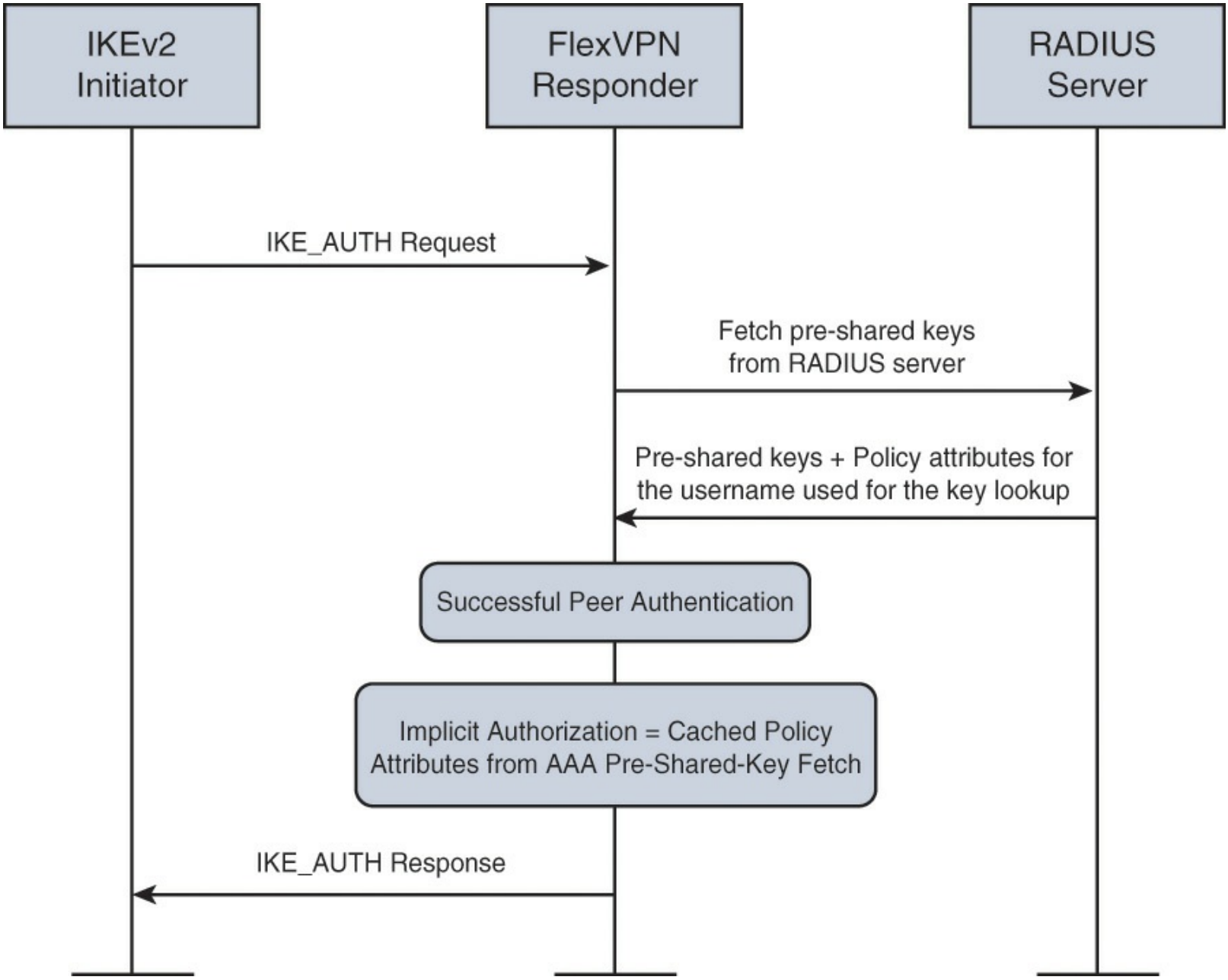


Figure 8-6 *FlexVPN Implicit Authorization with AAA-based, Pre-shared Keys*

FlexVPN Implicit Authorization Example

The following example illustrates FlexVPN implicit authorization configuration when a RADIUS server is used for storing pre-shared keys.

Step 1. Enable AAA.

```
aaa new-model
```

Step 2. Define RADIUS server group.

Click here to view code image

```
aaa group server radius radius_group1
 server name radius_server1
```

Step 3. Define the AAA method lists for authorization, using an external AAA server.

Click here to view code image

```
aaa authorization network aaa_list1 group radius_group1
```

Step 4. Define the RADIUS server.

Click here to view code image

```
radius server radius_server1
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_server1_key
```

Step 5. Define the IKEv2 name mangler.

Click here to view code image

```
crypto ikev2 name-mangler name_mangler1
 fqdn hostname
```

Step 6. Configure AAA-based pre-shared keys and FlexVPN implicit authorization in the IKEv2 profile as highlighted in the following example:

Click here to view code image

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share
 authentication remote pre-share
 keyring aaa aaa_list1 name-mangler name_mangler1
 aaa authorization user psk cached
```

The following example illustrates the fetching of pre-shared keys from a AAA server, and the caching of unsolicited policy attributes received from the AAA server for FlexVPN implicit authorization, using the selected output from **debug crypto ikev2**, **debug aaa subsys**, and **debug radius authentication** logs. The relevant debug outputs are highlighted.

Note that the peer is authenticating, using the pre-shared key method, and is using an IKE identity of type FQDN with value **router1.domain1**.

Click here to view code image

```
IKEv2:(SESSION ID = 3,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 80753858A6D0BEAB - Responder SPI : EA3464A2957A6F65 Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST
IKEv2:(SESSION ID = 3,SA ID = 1):Searching policy based on peer's identity
'router1.domain1' of type 'FQDN'
IKEv2:(SESSION ID = 3,SA ID = 1):Peer's authentication method is 'PSK'
```

Fetch pre-shared keys from AAA server.

Click here to view code image

```
IKEv2:(SESSION ID = 3,SA ID = 1):Get peer's preshared key for router1.domain1
IKEv2-INTERNAL:Fetching shared secret from AAA
IKEv2-INTERNAL:Name-mangler 'name_mangler1' returning mangled-name 'router1'
for name-type 1, name-len 15, name_string router1.domain1

IKEv2-INTERNAL:Using AAA method list aaa_list1 for peer router1
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Password request sent

AAA SRV(0000000E): process author req
AAA SRV(0000000E): Author method=SERVER_GROUP radius_group1
RADIUS/ENCODE: Best Local IP-Address 172.16.1.2 for Radius-Server 172.16.1.3
RADIUS(0000000E): Send Access-Request to 172.16.1.3:1645 id 1645/2, len 131
```

```
RADIUS:  authenticator 11 DD F2 16 3C 32 7A 9F - 26 6E AA DA 92 8D 99 8D
RADIUS:  User-Name            [1]   9   "router1"
RADIUS:  User-Password        [2]   18  *
RADIUS:  Calling-Station-Id   [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco        [26]  60
RADIUS:   Cisco AVpair        [1]   54  "audit-session-id=L2L4AC1G102ZO2L4AC1G101Z
I1F401F4ZO3"
RADIUS:  Service-Type         [6]   6   Outbound                  [5]
RADIUS:  NAS-IP-Address       [4]   6   172.16.1.2
RADIUS(0000000E): Sending a IPv4 Radius Packet
RADIUS(0000000E): Started 5 sec timeout
RADIUS: Received from id 1645/2 172.16.1.3:1645, Access-Accept, len 172
RADIUS:  authenticator B0 26 56 33 0F 1D 11 07 - A2 BE 8F 63 DD 71 DB B1
RADIUS:  Vendor, Cisco        [26]  33
RADIUS:   Cisco AVpair        [1]   27  "ipsec:route-set=interface"
RADIUS:  Vendor, Cisco        [26]  30
RADIUS:   Cisco AVpair        [1]   24  "ipsec:route-accept=any"
RADIUS:  Vendor, Cisco        [26]  45
RADIUS:   Cisco AVpair        [1]   39  "ipsec:route-set=prefix 192.168.1.1/24"
RADIUS:  Vendor, Cisco        [26]  44
RADIUS:   Cisco AVpair        [1]   38  "ipsec:tunnel-password=pre_shared_key"
RADIUS(0000000E): Received from id 1645/2
AAA SRV(0000000E): protocol reply PASS for Authorization
AAA SRV(0000000E): Return Authorization status=PASS
```

Note that, along with the pre-shared key, other policy attributes are also received from the AAA server.

Click here to view code image

```
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received password response
IKEv2-INTERNAL:Received symmmetric password from radius server
IKEv2-INTERNAL:Received cached attributes:
route-set-interface: 1, route-accept any tag:1 distance:1,

IKEv2:(SESSION ID = 3,SA ID = 1):Verification of peer's authentication data PASSED
```

FlexVPN Authorization Types: Co-existence and Precedence

The three FlexVPN authorization types namely user, group, and implicit authorization, can coexist in the IKEv2 profile and be used to fetch policy attributes for an IKEv2/IPsec session. The policy attributes obtained through each of the authorization type are merged and then applied to the session. The precedence between the authorization types comes into play when the same attribute is obtained from multiple authorization types. Figure 8-7 illustrates how authorization attributes from the different FlexVPN authorization types are merged.

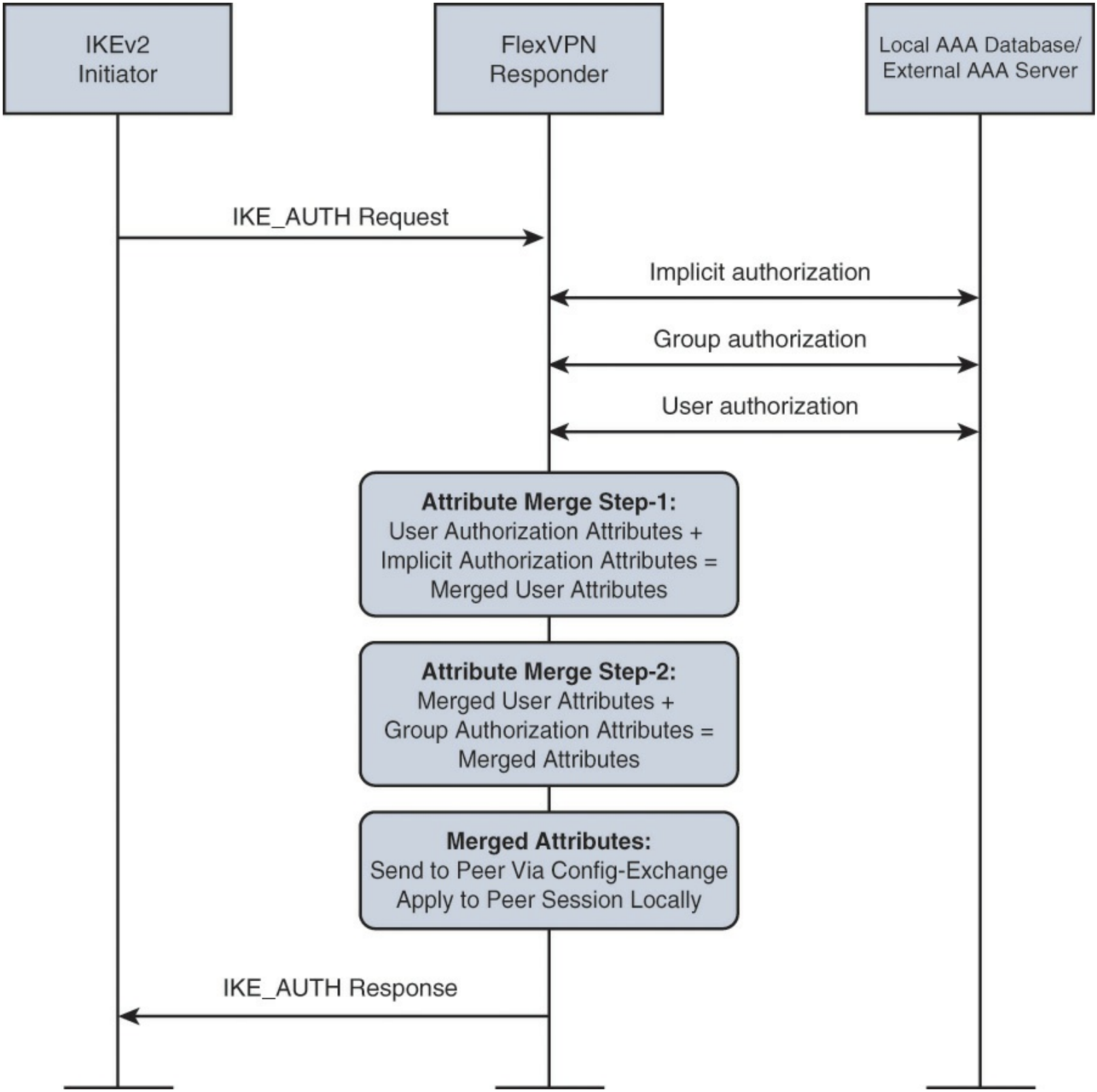


Figure 8-7 *Merging FlexVPN Authorization Attributes*

Following are the steps for merging the attributes and the precedence between the authorization types.

Step 1. The user authorization attributes are merged with the implicit authorization attributes, and the former takes precedence in case of duplicate attributes. The resulting attributes are called merged user attributes.

Step 2a. The merged user attributes are then combined with group authorization attributes, and the former takes precedence in case of duplicate attributes.

Step 2b. In Step 2a, the group authorization attributes can be configured to take precedence over the merged user attributes, using the **aaa authorization group override** command.

Step 3. The final merged attributes from Step 2 are sent to the peer through configuration exchange and/or are applied to the peer session.

User Authorization Taking Higher Precedence

The following example illustrates the merging of attributes from FlexVPN user authorization, group authorization and implicit authorization. The user authorization attributes take precedence over group authorization attributes, and this is the default behavior. IKEv2 name manglers are used to derive the name string for each of the authorization types. The AAA-based pre-shared-key feature is used for implicit authorization and configured to use the peer FQDN identity string for key lookup. User authorization is configured to use an external AAA server and the hostname portion of the peer FQDN identity string. Group authorization is configured to use a local AAA database and the domain portion of the peer FQDN identity string.

Step 1. Enable AAA and configure AAA authorization method lists, the AAA server group, and the RADIUS server.

Click here to view code image

```
aaa new-model

aaa group server radius radius_group1
 server name radius_server1

aaa authorization network aaa_psk_list group radius_group1
aaa authorization network user_author_list group radius_group1
aaa authorization network group_author_list local

radius server radius_server1
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_server1_key
```

Step 2. Configure the IKEv2 name manglers.

Click here to view code image

```
crypto ikev2 name-mangler aaa_psk_name_mangler
 fqdn all

crypto ikev2 name-mangler user_author_name_mangler
 fqdn hostname

crypto ikev2 name-mangler group_author_name_mangler
 fqdn domain
```

Step 3. Configure IKEv2 authorization policy for group authorization.

Click here to view code image

```
crypto ikev2 authorization policy domain1
 pool pool3
 def-domain domain1
```

Step 4. Enable AAA-based pre-shared-keys, and user, group, and implicit authorizations.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote any
```

```
 authentication local pre-share
 authentication remote pre-share
 keyring aaa aaa_psk_list name-mangler aaa_psk_name_mangler
 aaa authorization group psk list group_author_list name-mangler
group_author_name_mangler
 aaa authorization user psk cached
 aaa authorization user psk list user_author_list name-mangler
user_author_name_mangler
```

The selected output from **debug crypto ikev2** and **debug crypto ikev2 internal** logs illustrates the attribute merging process. The relevant output is highlighted.

- Pre-shared key is fetched from the AAA server and the unsolicited attributes received are cached.

Click here to view code image

```
IKEv2-INTERNAL:Fetching shared secret from AAA
IKEv2-INTERNAL:Using AAA method list aaa_psk_list for peer router1.domain1
IKEv2-INTERNAL:Received symmmetric password from radius server
IKEv2-INTERNAL:Received cached attributes:
ipv4-pool: pool1, route-set-interface: 1,
```

- Group authorization attributes are received.

Click here to view code image

```
IKEv2:Using mlist group_author_list and username domain1 for group author request
IKEv2-INTERNAL:Received group author attributes:
ipv4-pool: pool3, def-domain: domain1, route-accept any tag:1 distance:1,
```

- User authorization attributes are received.

Click here to view code image

```
IKEv2:Using mlist user_author_list and username router1 for user author request
IKEv2-INTERNAL:Received user author attributes:
ipv4-pool: pool2, route-accept any tag:1 distance:1,
```

- User authorization attributes are merged with implicit authorization attributes. Note that the **ipv4-pool** attribute is received in both user and implicit authorization, and the one from user authorization takes precedence and overrides the other.

Click here to view code image

```
IKEv2-INTERNAL:Merging and overriding cached attributes with user attributes
IKEv2-INTERNAL:Merged user attributes:
ipv4-pool: pool2, route-set-interface: 1, route-accept any tag:1 distance:1,
```

- Merged user authorization attributes are merged with group authorization attributes. Note that the **ipv4-pool** attribute is available in both, and the one from user authorization takes precedence and overrides the other.

Click here to view code image

```
IKEv2-INTERNAL:Merging and overriding group attributes with user attributes
IKEv2-INTERNAL:Merged attributes:
ipv4-pool: pool2, def-domain: domain1, route-set-interface: 1, route-accept any
tag:1 distance:1,
```

Group Authorization Taking Higher Precedence

The following example illustrates the merging of attributes from FlexVPN user authorization, group authorization and implicit authorization. The group authorization attributes are configured to take precedence over the user authorization attributes using the **aaa authorization group** command with the **override** option. The other configuration remains the same as in the previous example.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share
 authentication remote pre-share
 keyring aaa aaa_psk_list name-mangler aaa_psk_name_mangler
 aaa authorization group override psk list group_author_list name-mangler
group_author_name_mangler
 aaa authorization user psk cached
 aaa authorization user psk list user_author_list name-mangler
user_author_name_mangler
```

The selected output from **debug crypto ikev2** and **debug crypto ikev2 internal** logs illustrates the attribute merging process. The relevant output is highlighted.

- Pre-shared key is fetched from AAA server, and the unsolicited attributes received are cached.

Click here to view code image

```
IKEv2-INTERNAL:Fetching shared secret from AAA
IKEv2-INTERNAL:Using AAA method list aaa_psk_list for peer router1.domain1
IKEv2-INTERNAL:Received symmmetric password from radius server
IKEv2-INTERNAL:Received cached attributes:
ipv4-pool: pool1, route-set-interface: 1,
```

- Group authorization attributes are received.

Click here to view code image

```
IKEv2:Using mlist group_author_list and username domain1 for group author request
IKEv2-INTERNAL:Received group author attributes:
ipv4-pool: pool3, def-domain: domain1, route-accept any tag:1 distance:1,
```

- User authorization attributes are received.

Click here to view code image

```
IKEv2:Using mlist user_author_list and username router1 for user author request
IKEv2-INTERNAL:Received user author attributes:
ipv4-pool: pool2, route-accept any tag:1 distance:1,
```

- User authorization attributes are merged with implicit authorization attributes. Note that the **ipv4-pool** attribute is received in both and the one from user authorization takes precedence and overrides the other.

Click here to view code image

```
IKEv2-INTERNAL:Merging and overriding cached attributes with user attributes
IKEv2-INTERNAL:Merged user attributes:
ipv4-pool: pool2, route-set-interface: 1, route-accept any tag:1 distance:1,
```

- Merged user authorization attributes are merged with group authorization attributes. Note that the **ipv4-pool** attribute is available in both and the one from group authorization takes

precedence and overrides the other.

Click here to view code image

```
IKEv2-INTERNAL:Merging and overriding user attributes with group attributes
IKEv2-INTERNAL:Merged attributes:
ipv4-pool: pool3, def-domain: domain1, route-set-interface: 1, route-accept any
tag:1 distance:1,
```

FlexVPN Configuration Exchange

The IKEv2 protocol provides a mechanism for IKEv2 peers to exchange configuration information, using the configuration payload (CP) that is typically carried in the IKE_AUTH and INFORMATIONAL exchanges. The configuration payload can be of type CFG_REQUEST, CFG_REPLY, CFG_SET, and CFG_ACK and carries configuration attributes. The CFG_REQUEST and CFG_REPLY pair allows an IKE endpoint to request information from its peer, and the CFG_SET and CFG_ACK pair allows an IKE endpoint to push configuration data to its peer.

The configuration payloads are mainly used by remote-access clients to request an IP address from the remote-access server and other parameters such as DNS server, DHCP server, and so forth. The payloads are also used by IKEv2 nodes to exchange vendor-specific configuration attributes and to exchange the subnets protected by each node that determine which traffic must go through the IPsec tunnel and which traffic must not.

Enabling Configuration Exchange

Table 8-11 lists the IKEv2 profile commands that enable configuration exchange.

IKEv2 Profile Command	Description
config-exchange request	Enables sending of CFG_REQUEST by IKE initiators. IKE initiators include CFG_REQUEST in IKE_AUTH request and responders include CFG_REPLY in IKE_AUTH reply. This command is enabled by default.
config-exchange set send	Enables sending CFG_SET by IKE initiator and responder. CFG_SET is typically sent in the informational exchange. IKE initiators use CFG_SET to push information to the peer as CFG_REQUEST cannot be used to push information. IKE responders use CFG_SET to push information if CFG_REQUEST is not received, and hence information cannot be sent in CFG_REPLY. This command is enabled by default.
config-exchange set accept	Enables acceptance of CFG_SET received from a peer and sending of CFG_ACK. If this command is disabled, the received CFG_SET is ignored. This command is enabled by default.

Table 8-11 *Enabling Configuration Exchange*

FlexVPN Usage of Configuration Payloads

Table 8-12 lists the IKEv2 configuration payload types and their usage in FlexVPN. Note that the configuration payloads are not sent when crypto maps are used.

IKEv2 Configuration Payload Type	Usage in FlexVPN
CFG_REQUEST	Sent only by FlexVPN initiator when either of the following is true: ■ The **config-exchange request** command is configured in the IKEv2 profile. ■ Initiator is a FlexVPN client.
CFG_REPLY	Sent only by FlexVPN responder in reply to a CFG_REQUEST.
CFG_SET	Sent by FlexVPN initiator when all of the following are true: ■ The **config-exchange set send** command is configured in the IKEv2 profile. ■ The initiator has configuration data to send. Sent by FlexVPN responder when all of the following are true: ■ The **config-exchange set send** command is configured in the IKEv2 profile. ■ The responder has configuration data to send. ■ The responder did not receive a CFG_REQUEST.
CFG_ACK	Sent by the FlexVPN initiator and responder when all of the following are true: ■ CFG_SET is received from the peer. ■ The **config-exchange set accept** command is configured in the IKEv2 profile.

Table 8-12 *FlexVPN Usage of IKEv2 Configuration Payload Types*

Figure 8-8 illustrates FlexVPN's usage of CFG_REQUEST and CFG_REPLY to push configuration data from FlexVPN responder to an IKEv2 initiator. The merged attributes from FlexVPN authorizations source the values for the configuration attributes sent in the CFG_REPLY.

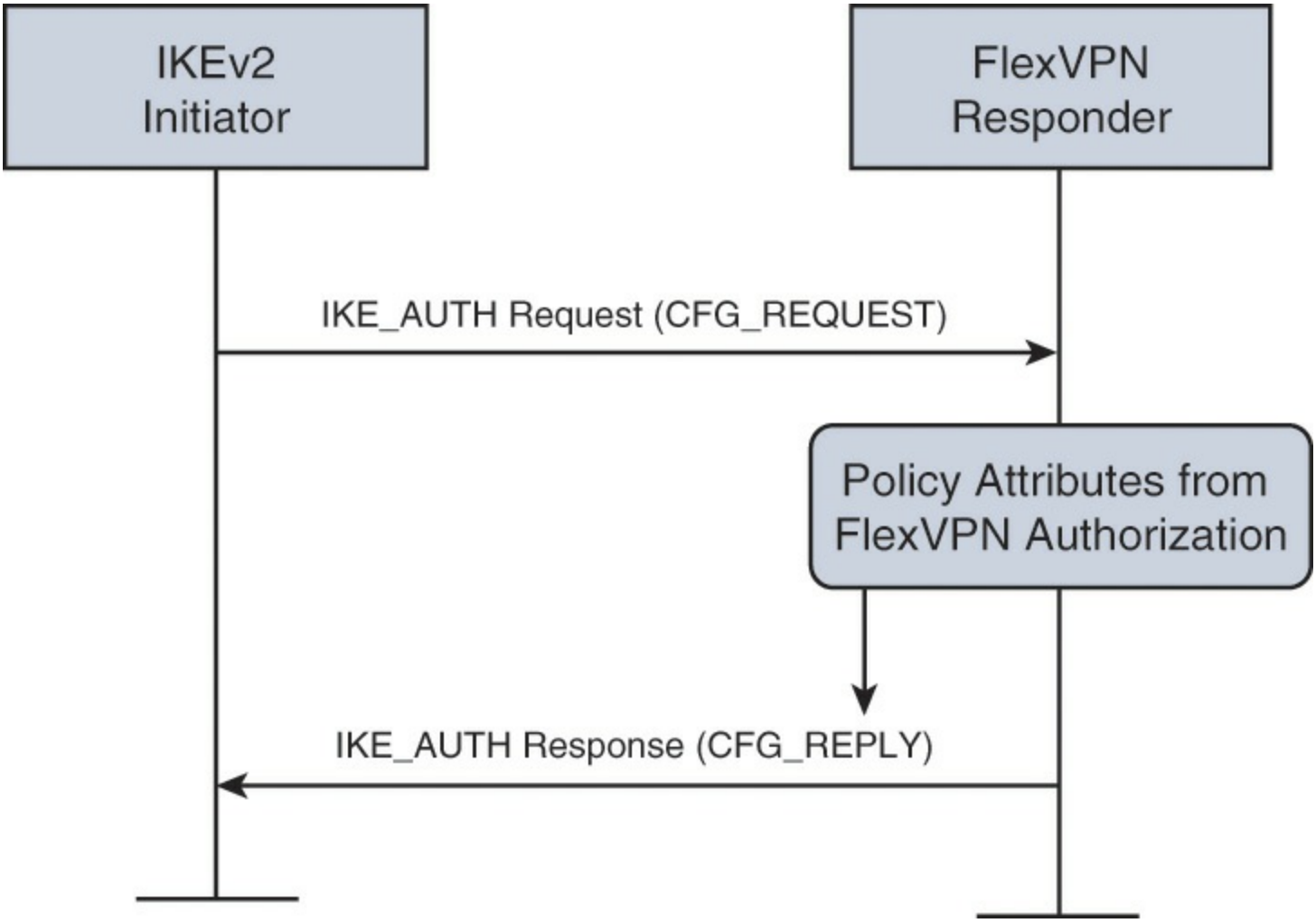


Figure 8-8 *Configuration Exchange, using CFG_REQUEST and CFG_REPLY*

Figure 8-9 illustrates FlexVPN's usage of CFG_SET and CFG_ACK to push configuration data from FlexVPN responder to an IKEv2 initiator when CFG_REQUEST is not received. The merged attributes from FlexVPN authorizations source the values for the configuration attributes sent in the CFG_SET in an informational exchange.

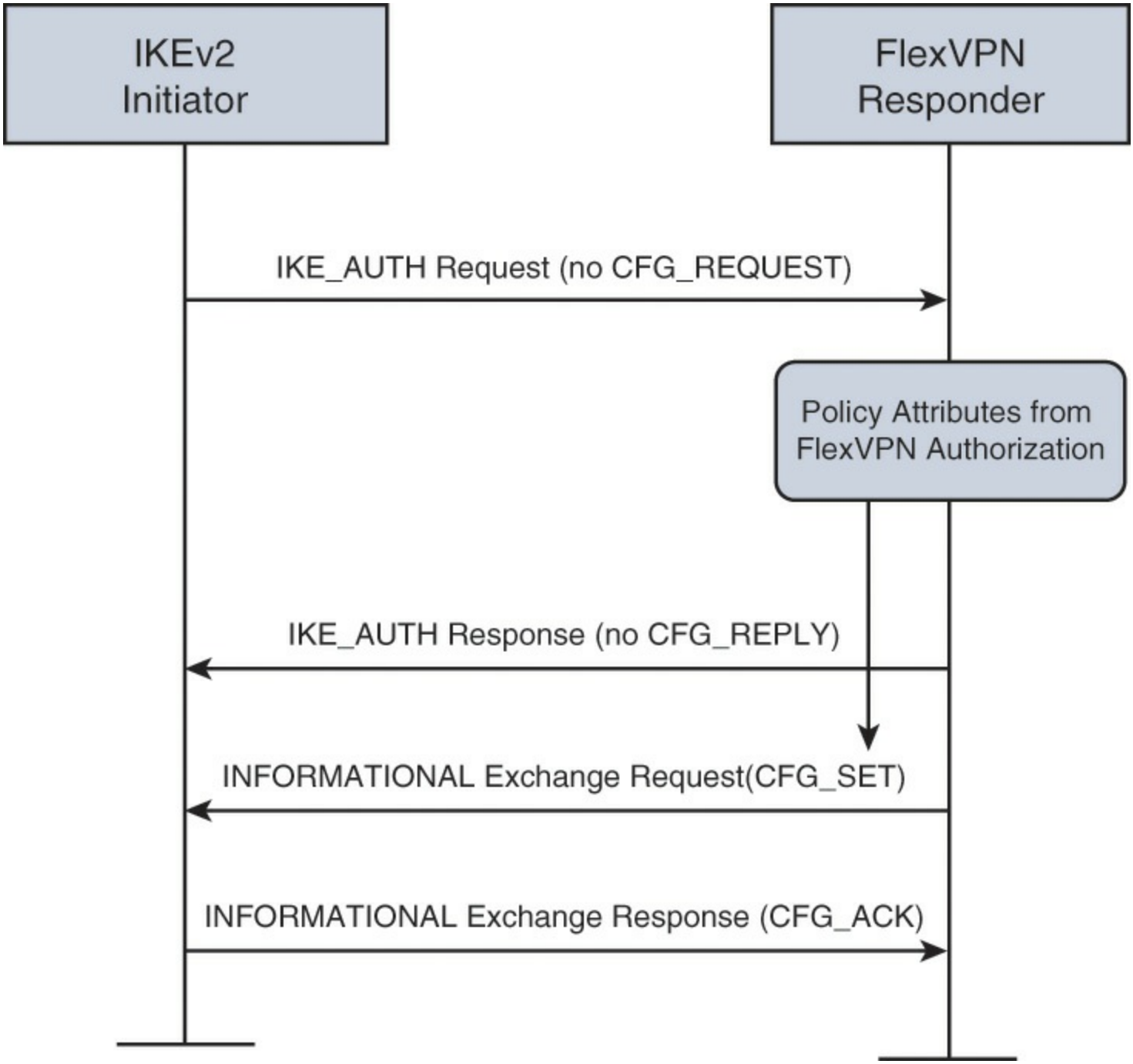


Figure 8-9 *Configuration Exchange, using CFG_SET and CFG_ACK*

Configuration Attributes and Authorization

The information in the configuration payload is carried in the form of TLV (type, length, value) structures known as *configuration attributes*. Table 8-13 lists the categories of configuration attributes supported by FlexVPN.

Attribute Category	Description
IKEv2 standard attributes	These are the attributes defined in the IKEv2 RFC.
Cisco unity attributes	These are Cisco proprietary attributes that are used between the FlexVPN server and the Cisco IKEv2 remote-access clients, such as AnyConnect client and FlexVPN client.
FlexVPN attributes	These are Cisco proprietary attributes that are used between the FlexVPN server and the FlexVPN client.

Table 8-13 *FlexVPN-Supported Configuration Attributes*

FlexVPN derives the values for most of the configuration attributes sent in CFG_REPLY and CFG_SET payloads from the AAA attributes obtained through FlexVPN authorization. The final merged AAA attributes from the FlexVPN authorization types are used to source the values for the configuration attributes.

Table 8-14 lists the IKEv2 standard configuration attributes supported by FlexVPN, their usage, and the corresponding AAA attributes from which their values are derived. The following table lists AAA attributes as RADIUS attributes, however most of the AAA attributes can be configured locally in the IKEv2 authorization policy as well. The RADIUS attributes supported by the FlexVPN server along with the corresponding IKEv2 authorization policy commands are listed in Chapter 9 - "FlexVPN Server".

Attribute	Description
INTERNAL_IP4_ADDRESS	Usage: This attribute is used by a remote-access server to assign an IPv4 address to a remote-access client. When requested in CFG_REQUEST, this attribute must be included in the CFG_REPLY with the assigned IP address. On failure to allocate the IP address, the session is terminated with INTERNAL_ADDRESS_FAILURE error. Value: The value for this configuration attribute is derived from the following AAA attributes in the listed order of precedence. 1. Framed-IP-Address 2. Framed-Pool (Local IPv4 address pool) 3. IP address obtained from the DHCP server, specified by the AAA attributes **group-dhcp-server**, **dhcp-giaddr**, **dhcp-timeout**
INTERNAL_IP4_NETMASK	Usage: This attribute is used along with the INTERNAL_IP4_ADDRESS attribute that defines the subnet on the remote-access server side. The traffic destined to it must be forwarded through the IPsec tunnel by the remote-access client.

	Value: The value is derived from the AAA attribute, Framed-IP-Netmask.
INTERNAL_IP6_ADDRESS	Usage: This attribute is used by a remote-access server to assign an IPv6 address to a remote-access client. When requested in CFG_REQUEST, this attribute must be included in the CFG_REPLY with the assigned IP address. On failure to allocate the IP address, the session is terminated with INTERNAL_ADDRESS_FAILURE error. Value: The value for this configuration attribute is derived from the following AAA attributes in the listed order of precedence. 1. **addrv6** and **prefix-len** 2. **ipv6-addr-pool** and **prefix-len**
INTERNAL_IP4_SUBNET	Usage: This attribute specifies an IPv4 subnet protected by the sending IKEv2 node and the receiving IKEv2 node must forward the traffic destined to this subnet through the IPsec tunnel. When there are multiple IPv4 subnets, this attribute is included multiple times in the CFG_REPLY or CFG_SET one for each IPv4 subnet. Value:

	The value for this configuration attribute is derived from the IPv4 subnets specified in all of the following AAA attributes ■ **route-set=interface [***interface***]** If an interface is not specified, the IP subnet is derived from the tunnel interface to which the IPsec session is attached. ■ **route-set=access-list** *acl-num | acl-name* ■ **route-set=prefix** *prefix/length* ■ **route-set=remote ipv4** *addr mask* Note that there can be one or more **route-set** attributes of each type.
INTERNAL_IP6_SUBNET	Usage: This attribute defines an IPv6 subnet protected by the sending IKEv2 node, and the receiving IKEv2 node must forward the traffic destined to this subnet through the IPsec tunnel. When there are multiple IPv6 subnets, this attribute is included multiple times in the CFG_REPLY or CFG_SET, one for each IPv6 subnet. Value:

	The value for this configuration attribute is derived from the IPv6 subnets specified in the all of the following AAA attributes ■ **route-set=interface [*interface*]** If interface is not specified, the IPv6 subnet is derived from the tunnel interface to which the IPsec session is attached. ■ **route-set=access-list ipv6 *acl-name*** ■ **route-set=remote ipv6 *prefix/length*** There can be one or more **route-set** attributes of each type.
INTERNAL_IP4_DNS	Usage: This attribute is used by a remote-access server to push an IPv4 address of a DNS server to a remote-access client. Value: The value is derived from the AAA attribute, **dns-servers**. There can be one or more such attributes.
INTERNAL_IP6_DNS	Usage: This attribute is used by a remote access server to push an IPv6 address of a DNS server to a remote-access client. Value: The value is derived from the AAA attribute, **ipv6-dns-servers-addr**. There can be one or more such attributes.

INTERNAL_IP4_NBNS	Usage: This attribute is used by a remote-access server to push an IPv4 address of a NetBios Name Server (WINS) to a remote-access client. Value: The value is derived from the AAA attribute **IPsec:wins-servers**. There can be one or more such attributes.
APPLICATION_VERSION	Usage: This attribute is used to exchange the version or application information of the IPsec host in ASCII format. FlexVPN sends this attribute only to Cisco peers. Value: The value is derived from the **show version** command output.

Table 8-14 *FlexVPN-Supported Configuration Attributes*

Following are the Cisco private use configuration attributes supported by FlexVPN. These attributes are sent in CFG_REPLY and CFG_SET only to Cisco peers. The implementation of these attributes is client and feature specific

- Cisco unity attributes sent only to Cisco peers (FlexVPN client and AnyConnect)
  - MODECFG_BANNER
  - MODECFG_DEFDOMAIN
  - MODECFG_SPLITDNS_NAME
  - MODECFG_INCLUDE_LOCAL_LAN
  - MODECFG_PFS
  - MODECFG_BACKUPSERVERS
  - MODECFG_SMARTCARD_REMOVAL_DISCONNECT
  - MODECFG_SPLIT_EXCLUDE
- Attribute sent only to FlexVPN client
  - MODECFG_CONFIG_URL
  - MODECFG_CONFIG_VERSION
- Attribute sent only to AnyConnect client
  - MODECFG_ANYCONNECT_SESSION_ID
  - MODECFG_ANYCONNECT_SESSION_DATA
  - MODECFG_ANYCONNECT_SESSION_TOKEN
  - MODECFG_ANYCONNECT_DPD_INTERVAL

- MODECFG_ANYCONNECT_CLEAN_UP_INTERVAL
- MODECFG_ANYCONNECT_IP4_MOS_ADDR
- MODECFG_ANYCONNECT_MOS_PORT
- MODECFG_RECONNECT_CLUSTER_IP

Following is the FlexVPN behavior with respect to configuration payloads and configuration attributes.

- FlexVPN responder includes INTERNAL_IP4_ADDRESS and INTERNAL_IP6_ADDRESS attributes along with the assigned IPv4/IPv6 addresses in CFG_REPLY only when these attributes are requested in CFG_REQUEST. These attributes are never sent unsolicited in CFG_REPLY.
- FlexVPN responder returns INTERNAL_ADDRESS_FAILURE if the requested addresses cannot be allocated.
- For attributes other than INTERNAL_IP4_ADDRESS and INTERNAL_IP6_ADDRESS requested in the CFG_REQUEST, FlexVPN responder includes the values for the requested attributes in the CFG_REPLY, if the requested attribute values are available through FlexVPN authorization. If not available, the attribute values are not included in the CFG_REPLY.
- FlexVPN responder includes the attribute values obtained through FlexVPN authorization in the CFG_REPLY, even if those attributes are not requested in received CFG_REQUEST. If CFG_REQUEST is not received, these attributes are sent through CFG_SET.
- FlexVPN supports dual stack with GRE encapsulation and not with native IPsec encapsulation. For clients that use dual stack over GRE such as FlexVPN and AnyConnect and that request both IPv4 and IPv6 configuration attributes in the CFG_REQUEST, both IPv4 and IPv6 attributes are included in the CFG_REPLY.
- For clients that use dual stack over native IPsec, such as Microsoft Windows IKEv2 client, and that request both IPv4 and IPv6 configuration attributes in CFG_REQUEST, either IPv4 or IPv6 attributes are included in the CFG_REPLY, depending on whether the negotiated IPsec Security Association traffic selectors are IPv4 or IPv6.
- FlexVPN nodes send Cisco private use configuration attributes and the standard APPLICATION_VERSION attribute only to Cisco peers determined by vendor ID.

Configuration Exchange Examples

The following example shows the exchange of configuration attributes between FlexVPN initiator and responder, using CFG_REQUEST, CFG_REPLY, CFG_SET, and CFG_SET. The relevant output from **debug crypto ikev2** and **debug crypto ikev2 internal** is highlighted.

Following is the configuration on initiator and responder. The IKEv2 profile has configuration exchange enabled by default and has group authorization configured, using a local AAA database.

Click here to view code image

```
aaa new-model

aaa authorization network default local

crypto ikev2 authorization policy default
 route set interface
```

```
 route accept any

crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 aaa authorization group psk list default default
 config-exchange set send
 config-exchange set accept
 config-exchange request
```

The following logs capture the CFG_REQUEST payload sent by the initiator in the IKE_AUTH request message.

Click here to view code image

```
IKEv2:Config data to send:
IKEv2:(SESSION ID = 1,SA ID = 1):Config-type: Config-request
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-dns, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-dns, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-nbns, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-nbns, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-subnet, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv6-dns, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv6-subnet, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: app-version, length: 277, data:
Cisco IOS Software, Linux Software (I86BI_LINUX-ADVENTERPRISEK9-M), Version
15.6(1.1)T,  ENGINEERING WEEKLY BUILD, synced to  V155_3_M0_2
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2015 by Cisco Systems, Inc.
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: split-dns, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: banner, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: config-url, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: backup-gateway, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: def-domain, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Have config mode data to send

IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 172.16.1.2:500/From
172.16.1.1:500/VRF i0:f0]
Initiator SPI : F66257AAE66F66A5 - Responder SPI : 6F177E021F80071F Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST
```

The following logs capture group authorization performed after peer authentication and the CFG_REPLY payload received by the initiator in the IKE_AUTH response message.

Click here to view code image

```
IKEv2:(SESSION ID = 1,SA ID = 1):Received Packet [From 172.16.1.2:500/To
172.16.1.1:500/VRF i0:f0]
Initiator SPI : F66257AAE66F66A5 - Responder SPI : 6F177E021F80071F Message id: 1
IKEv2 IKE_AUTH Exchange RESPONSE

IKEv2:(SESSION ID = 1,SA ID = 1):Verification of peer's authenctication data
PASSED
IKEv2-INTERNAL:IKEv2 local AAA author request for 'default'
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response
IKEv2-INTERNAL:VPN route info added for 'route set interface' attribute
IKEv2-INTERNAL:Received group author attributes:
route-set-interface: 1, route-accept any tag:1 distance:1,

IKEv2:Config data received:
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):Config-type: Config-reply
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-subnet, length: 8, data:
10.0.0.2 255.255.255.255
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: app-version, length: 277, data:
Cisco IOS Software, Linux Software (I86BI_LINUX-ADVENTERPRISEK9-M), Version
15.6(1.1)T,  ENGINEERING WEEKLY BUILD, synced to  V155_3_M0_2
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2015 by Cisco Systems, Inc.
```

The following logs capture the CFG_SET payload sent by the initiator in the INFORMATIONAL exchange request message. Note that the configuration attributes sent are derived from group authorization AAA attributes.

Click here to view code image

```
IKEv2-INTERNAL:Sending config-set
IKEv2:(SESSION ID = 1,SA ID = 1):Config-type: Config-set
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-subnet, length: 8, data:
10.0.0.1 255.255.255.255
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: app-version, length: 277, data:
Cisco IOS Software, Linux Software (I86BI_LINUX-ADVENTERPRISEK9-M), Version
15.6(1.1)T,  ENGINEERING WEEKLY BUILD, synced to  V155_3_M0_2
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2015 by Cisco Systems, Inc.

IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 172.16.1.2:500/From
172.16.1.1:500/VRF i0:f0]
Initiator SPI : F66257AAE66F66A5 - Responder SPI : 6F177E021F80071F Message id: 2
IKEv2 INFORMATIONAL Exchange REQUEST
```

The following logs capture the CFG_ACK payload received by the initiator in the INFORMATIONAL exchange reply message.

Click here to view code image

```
IKEv2:(SESSION ID = 1,SA ID = 1):Received Packet [From 172.16.1.2:500/To
172.16.1.1:500/VRF i0:f0]
Initiator SPI : F66257AAE66F66A5 - Responder SPI : 6F177E021F80071F Message id: 2
IKEv2 INFORMATIONAL Exchange RESPONSE

IKEv2:Config data received:
IKEv2:(SESSION ID = 1,SA ID = 1):Config-type: Config-ack
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-subnet, length: 0
```

The following logs capture group authorization performed after peer authentication and the CFG_REQUEST payload received by the responder in the IKE_AUTH request message.

Click here to view code image

```
IKEv2:(SESSION ID = 8,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 165DFC7D8B4DDE0A - Responder SPI : 61B417D0A611857A Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST

IKEv2:[Crypto Engine -> IKEv2] IKEv2 authentication data generation PASSED
IKEv2:Using mlist default and username default for group author request
IKEv2-INTERNAL:Received group author attributes:
route-set-interface: 1, route-accept any tag:1 distance:1,

IKEv2:Config data received:
IKEv2:(SESSION ID = 8,SA ID = 1):Config-type: Config-request
```

```
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-dns, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-dns, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-nbns, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-nbns, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-subnet, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv6-dns, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv6-subnet, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: app-version, length: 277, data:
Cisco IOS Software, Linux Software (I86BI_LINUX-ADVENTERPRISEK9-M), Version
15.6(1.1)T,  ENGINEERING WEEKLY BUILD, synced to  V155_3_M0_2
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2015 by Cisco Systems, Inc.
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: split-dns, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: banner, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: config-url, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: backup-gateway, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: def-domain, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Set received config mode data
```

The following logs capture the CFG_REPLY payload sent by the responder in the IKE_AUTH response message. Note that the configuration attributes sent are derived from group authorization AAA attributes.

Click here to view code image

```
IKEv2:Config data to send:
IKEv2:(SESSION ID = 8,SA ID = 1):Config-type: Config-reply
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-subnet, length: 8, data:
10.0.0.2 255.255.255.255
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: app-version, length: 277, data:
Cisco IOS Software, Linux Software (I86BI_LINUX-ADVENTERPRISEK9-M), Version
15.6(1.1)T,  ENGINEERING WEEKLY BUILD, synced to  V155_3_M0_2
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2015 by Cisco Systems, Inc.

IKEv2:(SESSION ID = 8,SA ID = 1):Sending Packet [To 172.16.1.1:500/From
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 165DFC7D8B4DDE0A - Responder SPI : 61B417D0A611857A Message id: 1
IKEv2 IKE_AUTH Exchange RESPONSE
```

The following logs capture the CFG_SET payload received by the responder in the INFORMATIONAL exchange request message.

Click here to view code image

```
IKEv2:(SESSION ID = 8,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 165DFC7D8B4DDE0A - Responder SPI : 61B417D0A611857A Message id: 2
IKEv2 INFORMATIONAL Exchange REQUEST

IKEv2:Config data received:
IKEv2:(SESSION ID = 8,SA ID = 1):Config-type: Config-set
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-subnet, length: 8, data:
10.0.0.1 255.255.255.255
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: app-version, length: 277, data:
Cisco IOS Software, Linux Software (I86BI_LINUX-ADVENTERPRISEK9-M), Version
15.6(1.1)T,  ENGINEERING WEEKLY BUILD, synced to  V155_3_M0_2
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2015 by Cisco Systems, Inc.
```

The following logs capture the CFG_ACK payload sent by the responder in the INFORMATIONAL

exchange reply message.

Click here to view code image

```
IKEv2:Config data to send:
IKEv2:(SESSION ID = 8,SA ID = 1):Config-type: Config-ack
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-subnet, length: 0

IKEv2:(SESSION ID = 8,SA ID = 1):Sending Packet [To 172.16.1.1:500/From
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 165DFC7D8B4DDE0A - Responder SPI : 61B417D0A611857A Message id: 2
IKEv2 INFORMATIONAL Exchange RESPONSE
```

The following example illustrates that when IKE responder has configuration data to send and CFG_REQUEST is not received, the data is sent, using CFG_SET. The relevant output from **debug crypto ikev2** and **debug crypto ikev2 internal** is highlighted.

Following is the configuration on initiator that has disabled the sending of a CFG_REQUEST.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote any
 identity local fqdn router1.domain1
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 aaa authorization group psk list default default
 no config-exchange request
```

The following logs capture group authorization performed after peer authentication and that no CFG_REQUEST payload is received by the responder in the IKE_AUTH request message.

Click here to view code image

```
IKEv2:(SESSION ID = 9,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 789338A336F1ACC2 - Responder SPI : 599AB29DACDBC6B5 Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST

IKEv2:(SESSION ID = 9,SA ID = 1):Verification of peer's authenctication data
PASSED
IKEv2:Using mlist default and username default for group author request
IKEv2-INTERNAL:Received group author attributes:
route-set-interface: 1, route-accept any tag:1 distance:1,
```

The following logs capture the CFG_SET payload sent by the responder in the INFORMATIONAL exchange request message. Note that the configuration attributes sent are derived from group authorization AAA attributes.

Click here to view code image

```
IKEv2-INTERNAL:Sending config-set
IKEv2:(SESSION ID = 9,SA ID = 1):Config-type: Config-set
IKEv2:(SESSION ID = 9,SA ID = 1):Attrib type: ipv4-subnet, length: 8, data:
10.0.0.2 255.255.255.255
IKEv2:(SESSION ID = 9,SA ID = 1):Attrib type: app-version, length: 277, data:
Cisco IOS Software, Linux Software (I86BI_LINUX-ADVENTERPRISEK9-M), Version
15.6(1.1)T,  ENGINEERING WEEKLY BUILD, synced to  V155_3_M0_2
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2015 by Cisco Systems, Inc.
```

```
IKEv2:(SESSION ID = 9,SA ID = 1):Sending Packet [To 172.16.1.1:500/From
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 789338A336F1ACC2 - Responder SPI : 599AB29DACDBC6B5 Message id: 0
IKEv2 INFORMATIONAL Exchange REQUEST
```

The following logs capture the CFG_ACK payload received by the responder in the INFORMATIONAL exchange reply message.

Click here to view code image

```
IKEv2:(SESSION ID = 9,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 789338A336F1ACC2 - Responder SPI : 599AB29DACDBC6B5 Message id: 0
IKEv2 INFORMATIONAL Exchange RESPONSE

IKEv2:Config data received:
IKEv2:(SESSION ID = 9,SA ID = 1):Config-type: Config-ack
IKEv2:(SESSION ID = 9,SA ID = 1):Attrib type: ipv4-subnet, length: 0
```

FlexVPN Routing

FlexVPN routing allows an IKEv2 node to automatically add routes to the subnets protected by remote IKEv2 peers in order to force traffic destined to those subnets through the IPsec tunnel. FlexVPN routes are static routes to remote subnets that point to the tunnel interface associated with the remote peer and are installed programmatically in the routing table.

One of the benefits of using FlexVPN routing is the ability to remove the need to run a routing protocol to advertise prefixes. Routing protocols incur control plane overhead and are chatty in nature; this can become costly when using low-powered products that are used in IoT or links that are charged on the amount of traffic sent.

FlexVPN routes are learned locally through FlexVPN authorization as well as from the peer through configuration exchange.

The remote subnets learned locally through FlexVPN authorization are derived from the following AAA attributes:

- **route-set = ipv4** ***address mask*** **next-hop** ***address*** **tag** ***tag*** **distance** ***distance***
- **route-set = ipv6** ***prefix/length*** **next-hop** ***address*** **tag** ***tag*** **distance** ***distance***

The remote subnets learned from the peer through configuration exchange are derived from the following configuration attributes:

- INTERNAL_IP4_SUBNET
- INTERNAL_IP4_NETMASK
- INTERNAL_IP4_SUBNET

Learning Remote Subnets Locally

The following example illustrates FlexVPN routes learned locally through FlexVPN authorization. The IKEv2 profile is configured with group authorization, using the local AAA database, which uses the IKEv2 authorization policy. The authorization policy is configured with the remote subnet, using the **route set local** command.

Click here to view code image

```
aaa new-model

aaa authorization network default local

crypto ikev2 authorization policy default
 route set local ipv4 192.168.6.0 255.255.255.0 tag 5000 distance 5
 route set interface

crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 aaa authorization group psk list default default
```

A single prefix is included in the route set local command above, if multiple prefixes are required this can easily be achieved by using the **route set access-list** command, which allows each access control entry in the ACL to specify a prefix.

The relevant output from **debug crypto ikev2** and **debug crypto ikev2 internal** is highlighted. The remote subnet is obtained through group authorization.

Click here to view code image

```
IKEv2:(SESSION ID = 1,SA ID = 1):Received Packet [From 172.16.1.2:500/To
172.16.1.1:500/VRF i0:f0]
Initiator SPI : 30DCC2A72A9F875A - Responder SPI : E2FB7F51411F1A5E Message id: 1
IKEv2 IKE_AUTH Exchange RESPONSE

IKEv2:(SESSION ID = 1,SA ID = 1):Verification of peer's authentication data PASSED
IKEv2-INTERNAL:IKEv2 local AAA author request for 'default'
IKEv2-INTERNAL:Received group author attributes:
route-set-interface: 1, route-set-local-ipv4: 192.168.6.0 255.255.255.0 tag 5000
distance 5 route-accept any tag:1 distance:1,
```

The remote subnet obtained through group authorization is programmatically installed as a static route in the routing table.

Click here to view code image

```
IKEv2-INTERNAL:VPN Route Added 192.168.6.0 255.255.255.0 via Tunnel0 in vrf global
```

The **`show ip route`** command output displays the installed route.

Click here to view code image

```
Router#show ip route 192.168.6.0
Routing entry for 192.168.6.0/24
  Known via "static", distance 5, metric 0 (connected)
  Tag 5000
  Routing Descriptor Blocks:
  * directly connected, via Tunnel0, permanent
      Route metric is 0, traffic share count is 1
      Route tag 5000
```

Learning Remote Subnets from Peer

The following example illustrates FlexVPN routes learned from the peer through configuration exchange. The IKEv2 profile is configured with group authorization, using the local AAA database pointing to the default IKEv2 authorization policy. It, in turn, is configured to accept CFG_SET using the **route accept** command.

Click here to view code image

```
aaa new-model

aaa authorization network default local

crypto ikev2 authorization policy default
 no route set interface
 route accept any

crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 aaa authorization group psk list default default
```

The relevant output from **debug crypto ikev2** and **debug crypto ikev2 internal** is highlighted.

Click here to view code image

```
IKEv2:(SESSION ID = 1,SA ID = 1):Received Packet [From 172.16.1.2:500/To
172.16.1.1:500/VRF i0:f0]
Initiator SPI : AE84121BC64D641B - Responder SPI : 89F6DA91AC2A2823 Message id: 1
IKEv2 IKE_AUTH Exchange RESPONSE

IKEv2:(SESSION ID = 1,SA ID = 1):Verification of peer's authentication data PASSED
IKEv2:Using mlist default and username default for group author request
IKEv2-INTERNAL:Received group author attributes:
route-accept any tag:1 distance:1,
```

The remote subnet information is received in the CFG_REPLY.

Click here to view code image

```
IKEv2-INTERNAL:Received config data from toolkit:
IKEv2:Config data received:
IKEv2:(SESSION ID = 1,SA ID = 1):Config-type: Config-reply
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-subnet, length: 8, data:
10.0.0.2 255.255.255.255
```

The remote subnet received in the CFG_REPLY is programmatically installed as a static route in the routing table.

Click here to view code image

```
IKEv2-INTERNAL:Subnet address: 10.0.0.2 255.255.255.255
IKEv2:VPN Route Added 10.0.0.2 255.255.255.255 via Tunnel0 in vrf global
```

The **show ip route** command output displays the installed route.

Click here to view code image

```
Router#show ip route
Gateway of last resort is not set
```

```
      4.0.0.0/24 is subnetted, 1 subnets
S        4.4.4.0 is directly connected, Tunnel0
      10.0.0.0/8 is variably subnetted, 3 subnets, 2 masks
S        10.0.0.2/32 is directly connected, Tunnel0

Router#show ip route 10.0.0.2
Routing entry for 10.0.0.2/32
  Known via "static", distance 1, metric 0 (connected)
  Tag 1
  Routing Descriptor Blocks:
  * directly connected, via Tunnel0, permanent
      Route metric is 0, traffic share count is 1
      Route tag 1
```

For maximum security, the prefixes installed for a certain peer can be restricted, using the **route set local** command. This ensures that a malicious or misconfigured peer does not install prefixes it does not own. Since CSCuw23906 has been integrated into version 16.2, if a client requests an IP address using configuration payload, the **no route accept** command can be applied on the gateway to ensure that a route to the IP address assigned to the virtual-access interface is applied. However, the peer cannot push any prefixes to the gateway.

Summary

This chapter introduced the Cisco IOS FlexVPN building blocks, such as the Cisco IOS point-to-point tunnel interface and AAA infrastructure. The chapter explained how FlexVPN leverages AAA to support user, group, and implicit authorizations, using the local and external AAA databases. How the IKEv2 name mangler is used to derive the AAA username for authorization based on the peer IKE identity was also covered. The chapter then explained how the FlexVPN authorization sources the data for configuration exchange and forms the basis for FlexVPN routing. The most important point to remember from this chapter is how FlexVPN leverages the point-to-point tunnel interfaces, AAA, and IKEv2 configuration exchange to support advanced features.

Chapter 9. FlexVPN Server

This chapter introduces the Cisco IOS FlexVPN server. The FlexVPN server acts as a VPN headend for the remote-access and hub-spoke VPN topologies. The FlexVPN server leverages the IKEv2 configuration and the FlexVPN building blocks discussed in the earlier chapters. It dynamically instantiates a point-to-point tunnel interface for every peer session and leverages AAA to support EAP authentication, AAA-based pre-shared keys, session accounting, and for authorization to derive session policy attributes that are sent to the peer via configuration exchange and/or applied on the session interface. The chapter discusses the features that are relevant to FlexVPN server.

The chapter covers the following main topics

- Sequence of events
- EAP authentication
- AAA-based pre-shared keys
- AAA Accounting
- Per session interface creation
- Auto detection of tunnel transport and encapsulation
- RADIUS packet of disconnect
- RADIUS change of authorization
- IKEv2 auto-reconnect
- User authentication using AnyConnect-EAP
- Dual-factor authentication using AnyConnect-EAP
- RADIUS attributes supported by the FlexVPN server
- Remote access clients supported by the FlexVPN server

Sequence of Events

Figure 9-1 illustrates the high-level sequence of events on FlexVPN server. Understanding the sequence of events helps one better understand the various features and troubleshoot any issues related to the FlexVPN server.

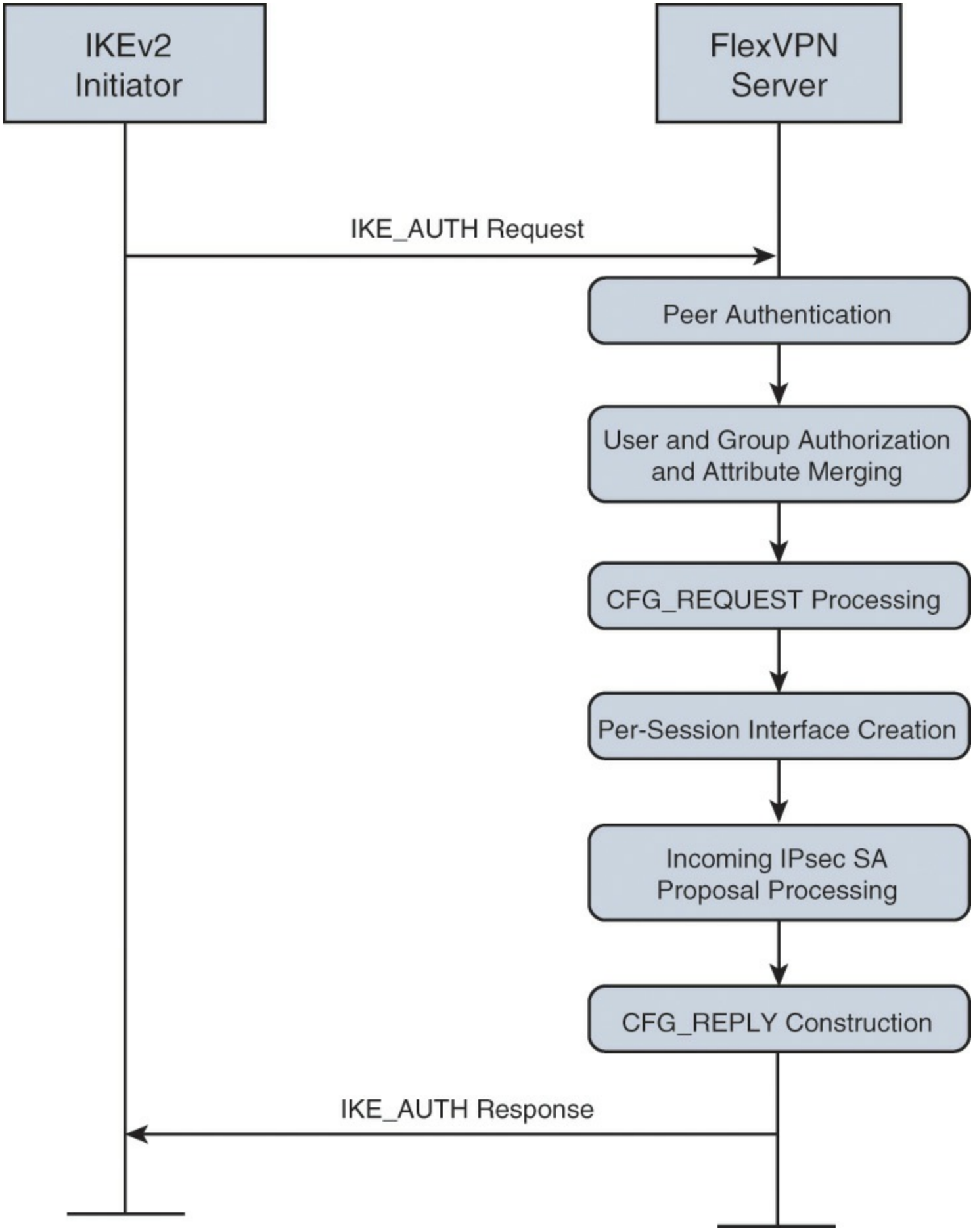


Figure 9-1 *The FlexVPN Server Sequence of Events*

A high-level description of the sequence of events on the FlexVPN server follows:

1. After successful peer authentication, the configured authorizations are performed, based on the

authenticated peer identity, and the attributes from the various authorization types are merged. If the group or user authorization fails, the session is terminated.

2. If the CFG_REQUEST payload is received, it is then processed to specifically check whether the peer has requested an IPv4 and/or IPv6 address. Either IPv4/IPv6 address or both addresses are allocated from the merged authorization attributes, depending on whether the IPsec session being negotiated is single or dual stacked. If the required addresses cannot be allocated, the session is terminated.
3. The virtual-access interface (P2P tunnel interface) is then instantiated for the incoming session, from the virtual-template configured in the IKEv2 profile. The configuration for the virtual-access interface is cloned from the virtual-template and from the AAA authorization attributes (**interface-config** attributes) as configured.
4. The incoming IPsec Security Association proposal is then processed, and if the proposal is acceptable, the IPsec Security Association is attached to the virtual-access interface and installed in the crypto engine and the data plane.
5. The CFG_REPLY payload is then constructed from the merged authorization attributes if the CFG_REQUEST payload was received.

EAP Authentication

The FlexVPN server supports EAP only as a remote authentication method, using an external EAP server, and does not support EAP as a local authentication method. In EAP terminology, the FlexVPN server acts a pass-through authenticator that relays EAP messages between an EAP server and the IKEv2 clients that, in turn, act as EAP supplicants/peers. When authenticating IKEv2 clients using EAP, the FlexVPN server must authenticate, using a certificate-based authentication method.

Figure 9-2 illustrates the use of EAP authentication in IKEv2 with the FlexVPN server. The FlexVPN server first authenticates, using a certificate-based method after which the IKEv2 client authenticates, using EAP. The IKEv2 client and the FlexVPN server then perform mutual authentication with the pre-shared key method, using the EAP shared secret if available or the Diffie-Hellman shared key-based SK_pi, and SK_pr as the pre-shared key.

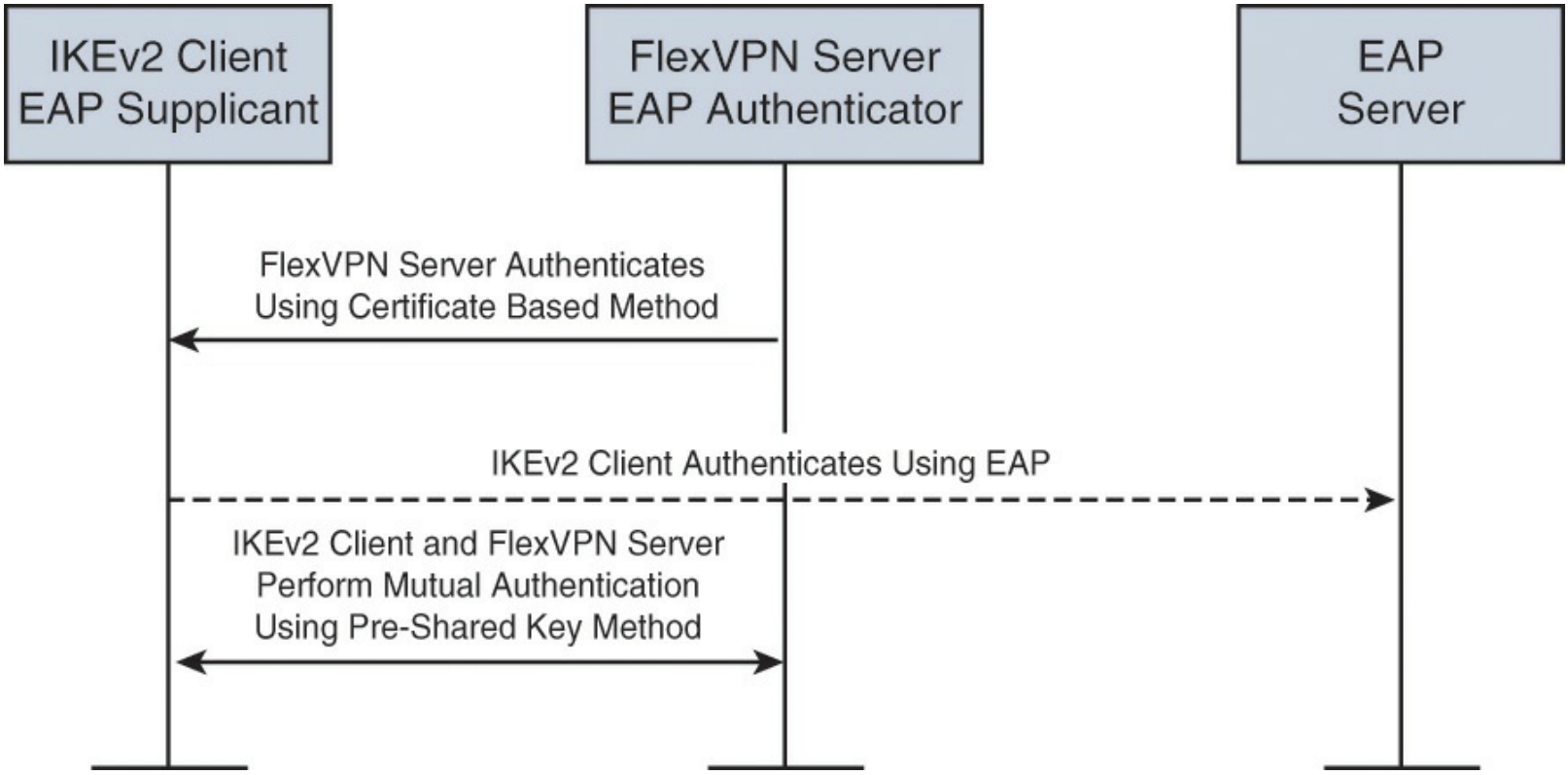


Figure 9-2 *EAP Authentication on the FlexVPN Server*

EAP Methods

FlexVPN server acts a pass-through authenticator. Thus, the EAP method negotiated and used between the IKEv2 client and the EAP server for authentication is transparent to the FlexVPN server. There are two types of EAP methods that are relevant to IKEv2 EAP authentication: key-generating methods and non key-generating methods.

The key-generating EAP methods create a shared key known as *master session key* (MSK) as a side effect of authentication. After successful authentication, the EAP server communicates this MSK to the FlexVPN server. The IKEv2 client and the FlexVPN server use this EAP shared secret (MSK) as a pre-shared key to authenticate each other in the final IKE_AUTH exchange.

The non key-generating EAP methods do not create a shared key as part of EAP authentication. In this case, the IKEv2 client and the FlexVPN server use SK_pi and SK_pr derived from the Diffie-Hellman shared secret as a pre-shared key to authenticate each other in the final IKE_AUTH exchange.

Note that the use of non key-generating EAP methods is not recommended as they are susceptible to man-in-the-middle attacks.

The EAP methods that use transport layer security (TLS), such as EAP-TLS or EAP-TTLS, will use additional cryptographic processing in addition to the cryptographic processing for IKEv2. EAP tunneled methods, such as EAP-TLS and EAP-PEAP, incur more processing than nontunneled methods.

EAP Message Flow

Figure 9-3 illustrates the EAP message flow between an IKEv2 client, the FlexVPN server, and the RADIUS-based EAP server.

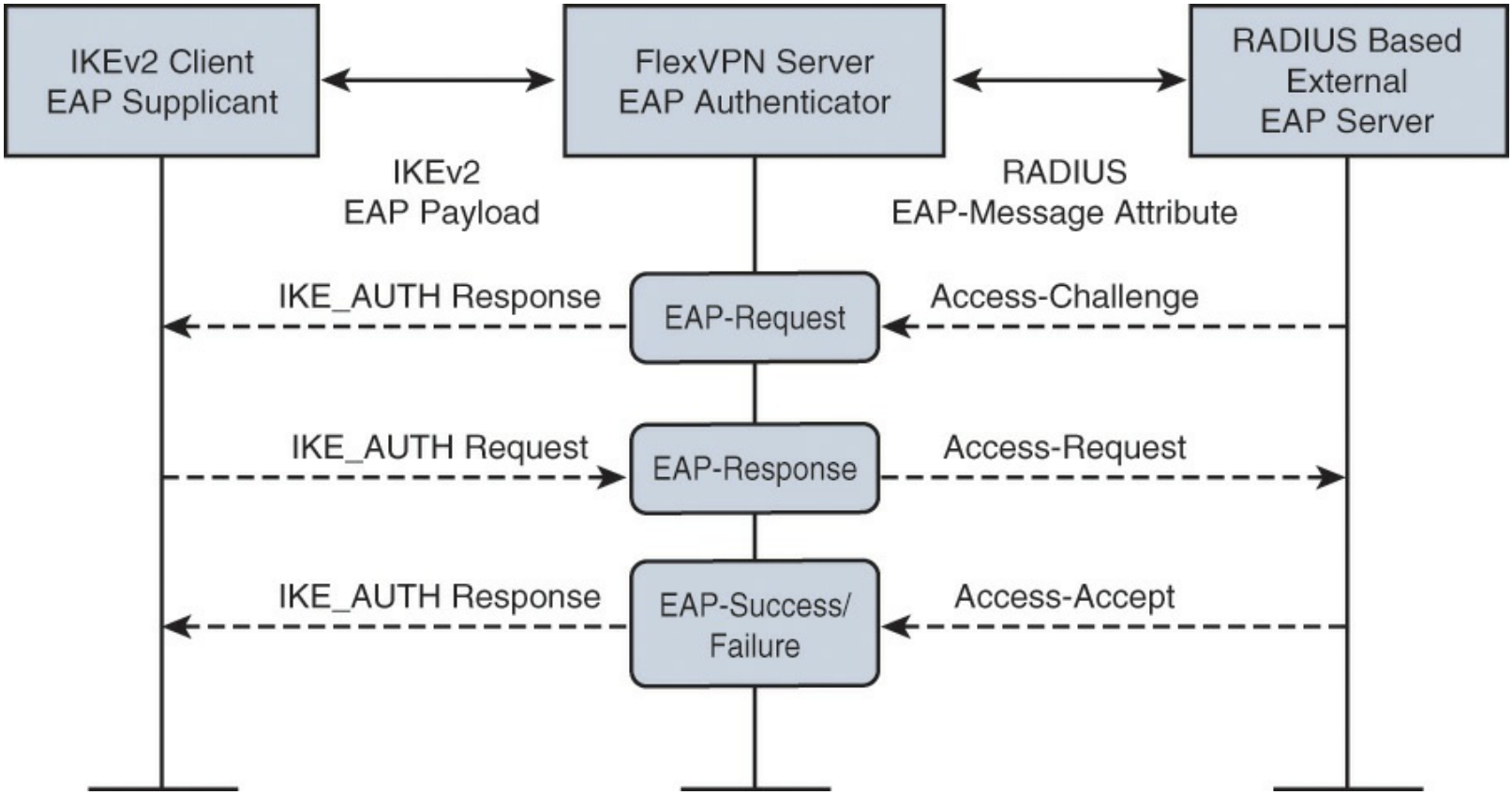


Figure 9-3 *EAP Message Flow*

- The EAP messages between the IKEv2 client and the FlexVPN server are embedded within the IKEv2 EAP payload and are carried in the IKE_AUTH request and response messages.
- The EAP messages between the FlexVPN server and the RADIUS-based EAP server are embedded within the RADIUS EAP-Message attribute and are carried in the RADIUS Access-Request, Access-Challenge, and Access-Accept messages.
- The FlexVPN server, acting as EAP authenticator, relays the EAP messages between the IKEv2 client and the EAP server by translating between the IKEv2 EAP payload and the RADIUS EAP-Message attribute.
- The EAP server uses EAP-Request message to request information from the IKEv2 client (EAP supplicant), and the IKEv2 client uses EAP-Response message to provide the requested information.
- The EAP server uses EAP-Success/EAP-Failure messages to convey the authentication result.

EAP Identity

Figure 9-4 illustrates the EAP identity that the FlexVPN server uses for EAP authentication and subsequent user and group authorization.

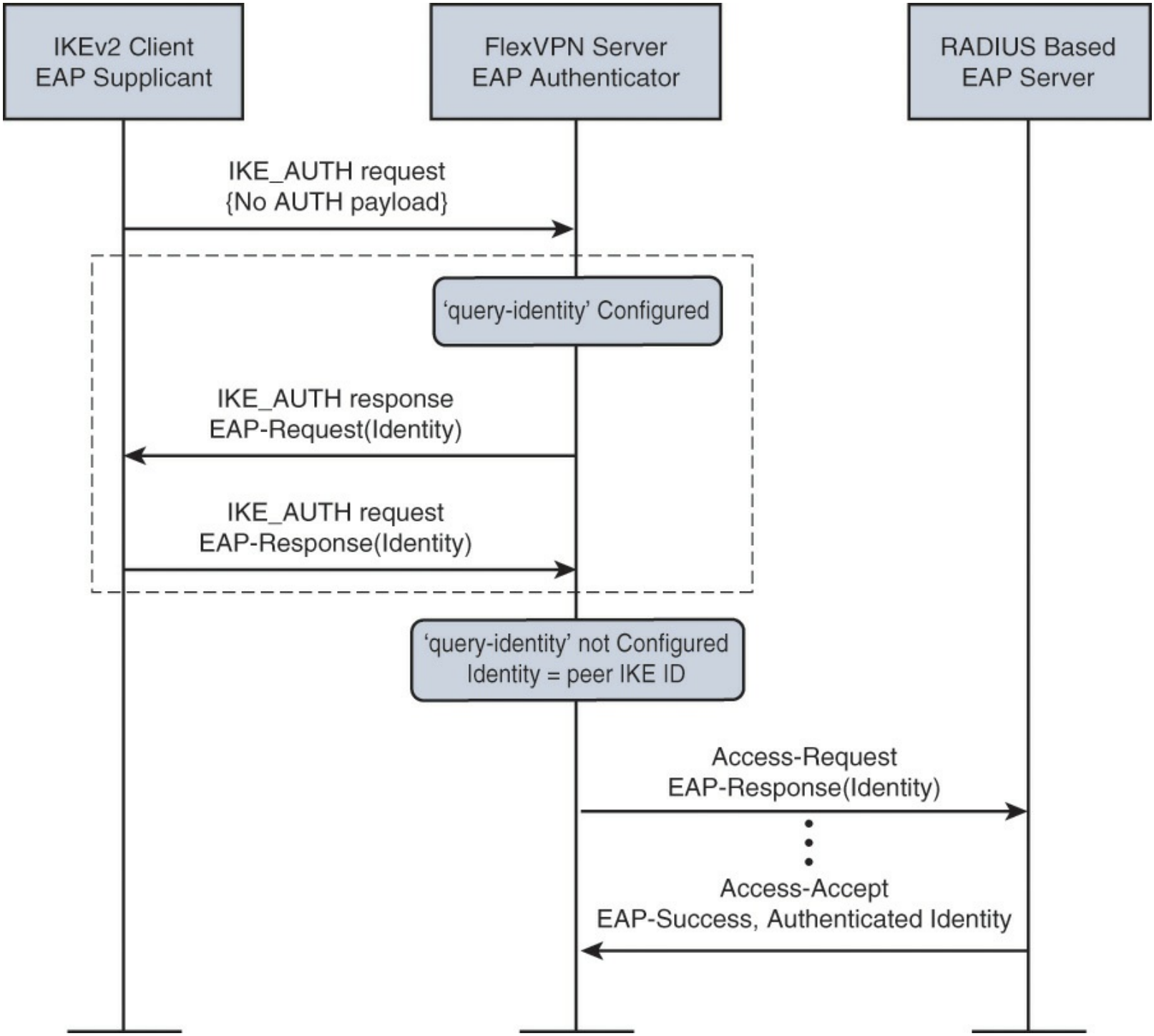


Figure 9-4 *EAP Identity for Authentication and Authorization*

- The FlexVPN server allows for querying EAP identity from the IKEv2 client, which is enabled using the **query-identity** configuration option. With the **query-identity** option, the FlexVPN server sends an EAP-Request message to the IKEv2 client, requesting the EAP identity. When the IKEv2 client sends an EAP-Response with its EAP identity, the FlexVPN server relays this EAP identity to the EAP server to kick off the EAP authentication. Note that some IKEv2 clients, such as AnyConnect and Microsoft Windows IKEv2 client, expect an EAP-identity request before any other EAP request. Thus, they require that the **query-identity** option be configured on the FlexVPN server.
- When the **query-identity** configuration option is not enabled, FlexVPN server uses the peer IKE identity as EAP identity and relays it to the EAP server. However, the peer IKE identity of type IP address is not supported as an EAP identity, and the negotiation is terminated if the peer IKE identity is of type IP address and the **query-identity** option is not enabled.

- After successful EAP authentication, the authenticated EAP identity returned by the EAP server, along with the EAP success message, is used to derive the AAA username for user and group authorizations. It should be noted that in some circumstances, the identity returned in the Access-Accept message will be different to that sent in the EAP exchange.

EAP Timeout

Depending on the EAP method, EAP authentication might involve prompting the user on the IKEv2 client side to enter EAP credentials of username and password. This can cause the IKEv2 client to take longer to respond to EAP request messages from the FlexVPN server, compared to other protocol exchanges. The FlexVPN server allows defining the duration for which FlexVPN server will wait for EAP-Response from the IKEv2 client before timing out and aborting the negotiation. This timeout value is configured using **timeout** option in the EAP authentication configuration in the IKEv2 profile. Figure 9-5 illustrates EAP timeout.

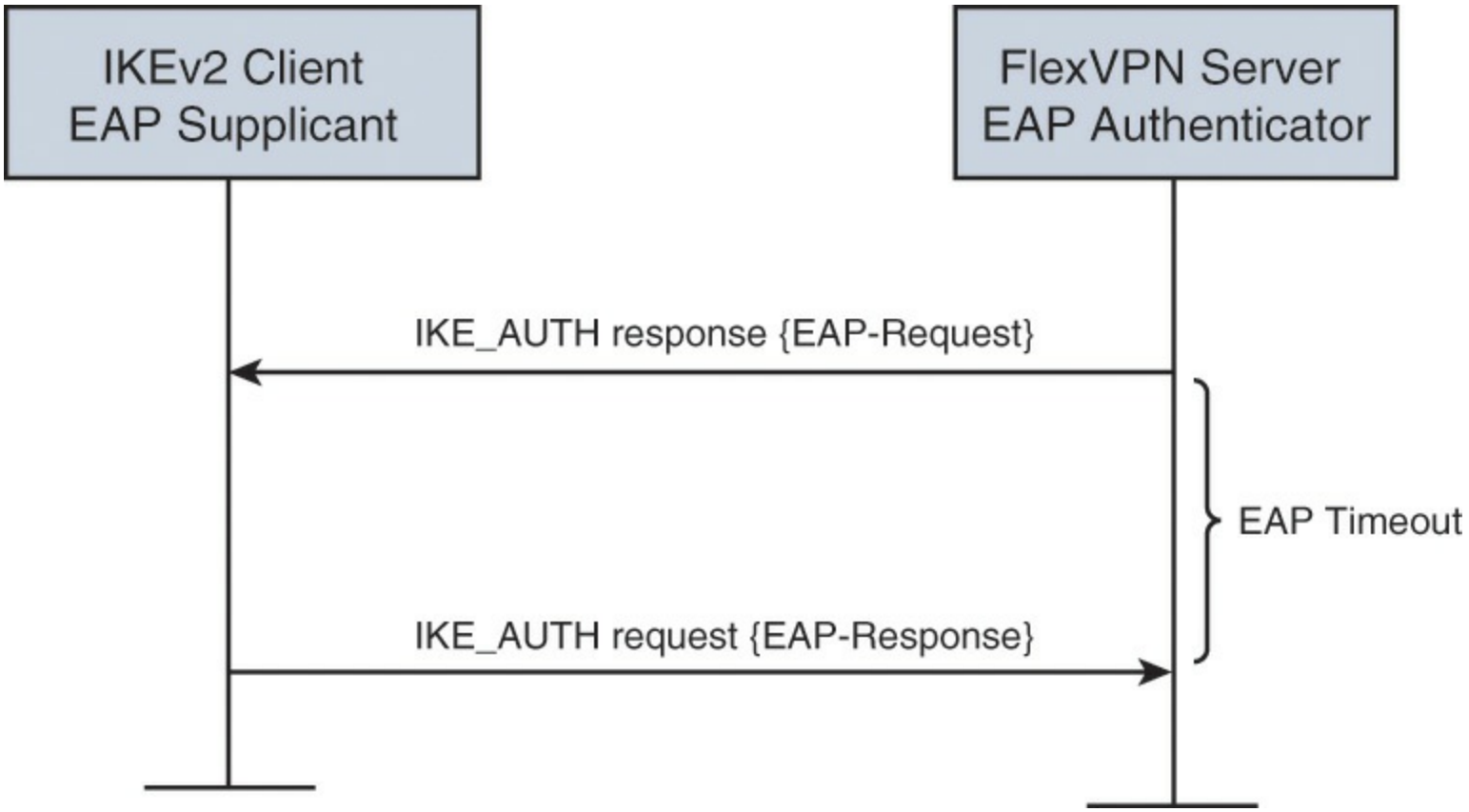


Figure 9-5 *EAP Timeout*

EAP Authentication Steps

Figure 9-6 illustrates the detailed steps involved when the FlexVPN server authenticates IKEv2 clients, using EAP with a RADIUS-based external EAP server.

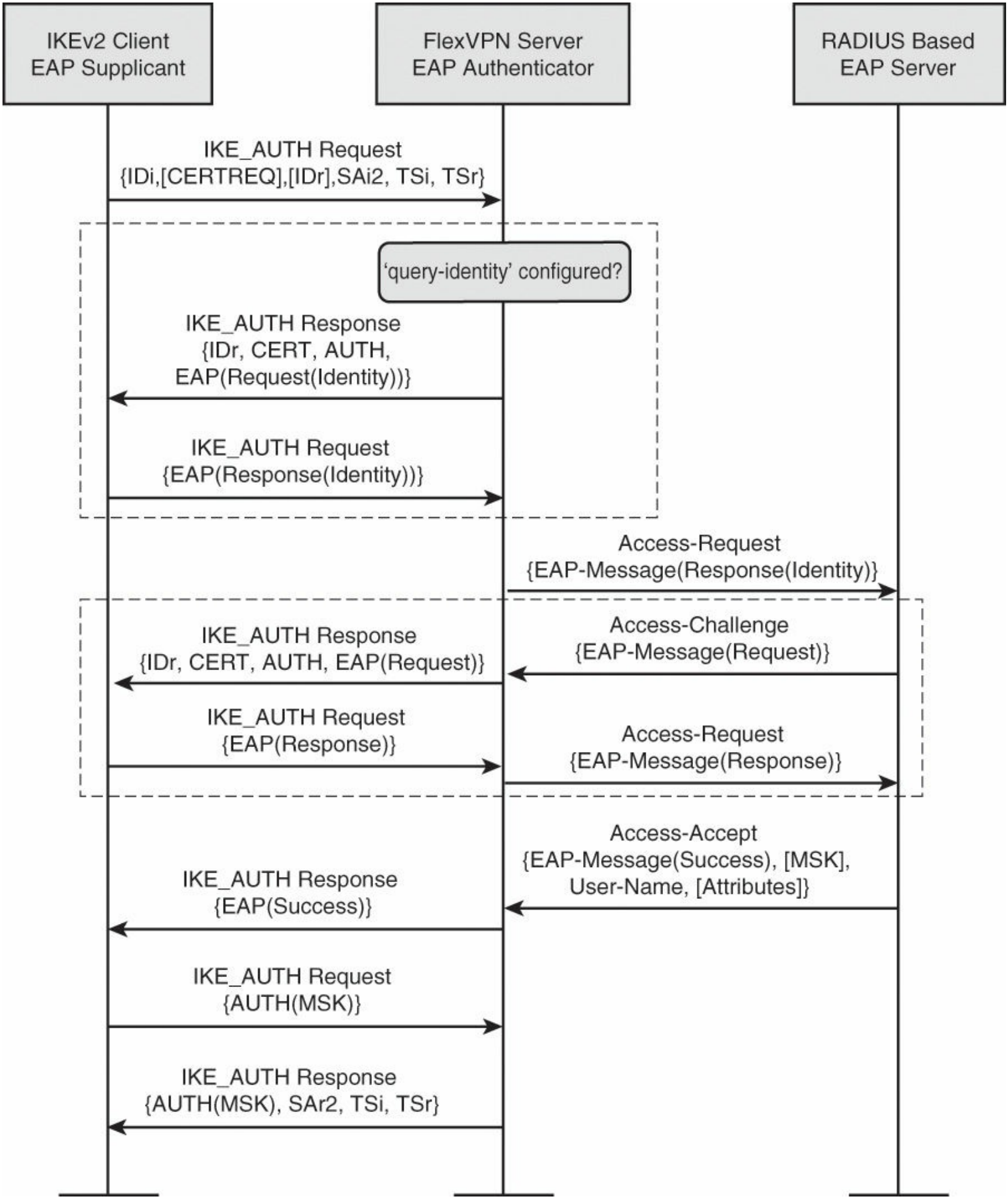


Figure 9-6 *FlexVPN Server EAP Authentication Steps*

The following is a description of the steps involved.

- The IKEv2 client indicates its intent to authenticate, using EAP, by leaving out the AUTH

payload in the IKE_AUTH request.

- If the FlexVPN server is configured to authenticate this client, using the EAP method with the **query-identity** option, the FlexVPN server sends an EAP-Request message to the IKEv2 client, requesting the EAP identity. On receiving the EAP identity from the IKEv2 client, FlexVPN server relays the EAP identity to the EAP server in an EAP-Response message.
- If the FlexVPN server is configured to authenticate this client, using an EAP method, but the **query-identity** option is not configured, the FlexVPN server relays the peer IKE identity as EAP identity to the EAP server in an EAP-Response message.
- The EAP server and the IKEv2 client then use EAP-Request and EAP-Response messages to exchange information, to negotiate the EAP method and to perform the method-specific authentication exchanges. The FlexVPN server relays the EAP-Request and EAP-Response messages between them. Note that the IDr, CERT and AUTH payloads would be sent only in the first IKE_AUTH response.
- The EAP server uses EAP-Success/EAP-Failure messages to convey the authentication result. If the EAP authentication is successful, the EAP server communicates the authenticated EAP identity and the master session key (MSK)-shared secret if a key-generating EAP method was used.
- The MSK is used to substitute the pre-shared key in computing the AUTH payload used in the final IKE_AUTH messages between the IKEv2 client and the FlexVPN server. In the absence of MSK, the SK_pi and SK_pr values derived from the Diffie-Hellman shared secret are used to substitute the pre-shared key.
- The authenticated EAP identity is used to derive the AAA username for user and group authorizations.
- If the EAP authentication is successful, the RADIUS server also pushes the policy attributes corresponding to the authenticated identity. These attributes can be configured to be cached and used, which is known as *implicit authorization*.

Configuring EAP

The following example illustrates configuring EAP as a remote authentication method on the FlexVPN server using a RADIUS-based external EAP server. EAP is configured in the IKEv2 profile.

In order to configure EAP as a remote authentication method, the local authentication method must be certificate-based.

Click here to view code image

```
Router(config-ikev2-profile)#authentication remote eap
For remote auth method to be EAP, the local auth method must be certificate-based
Router(config-ikev2-profile)#authentication local rsa-sig
Router(config-ikev2-profile)#authentication remote eap
```

The **query-identity** option makes the FlexVPN server query the EAP identity from the IKEv2 client and relay it to the EAP server.

Click here to view code image

```
Router(config-ikev2-profile)#authentication remote eap ?
  query-identity  query EAP identity from peer
```

```
  timeout          timeout
  <cr>
```

The **timeout** option defines the duration for which the FlexVPN server will wait for an EAP-Response from the IKEv2 client after sending the EAP-Request, before timing out. The default timeout is 90 seconds.

Click here to view code image

```
Router(config-ikev2-profile)#authentication remote eap timeout ?
  <45-180>  EAP authentication timeout in seconds
```

The AAA method list specifying the RADIUS server to use for EAP authentication is configured using the below command.

Click here to view code image

```
Router(config-ikev2-profile)#aaa authentication eap ?
  WORD  AAA list name
```

EAP Configuration Example

The following example illustrates sample EAP configuration and logs on the FlexVPN server for EAP as a remote authentication method using a RADIUS-based external EAP server

Step 1. Enable AAA.

```
aaa new-model
```

Step 2. Define the AAA authentication method list to specify the RADIUS server hosting the EAP server.

Click here to view code image

```
aaa authentication login authen_list_radius group radius_group
```

Step 3. Define the RADIUS server group.

Click here to view code image

```
aaa group server radius radius_group
 server name radius_server
```

Step 4. Define the RADIUS server.

Click here to view code image

```
radius server radius_server
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_key
```

Step 5. Configure EAP authentication in the IKEv2 profile.

Click here to view code image

```
crypto ikev2 profile ikev2_profile
 aaa authentication eap authen_list_radius
 authentication remote eap query-identity
 authentication remote eap timeout 120
 authentication remote rsa-sig
 aaa authentication eap authen_list_radius
```

The following **debug crypto ikev2**, **debug crypto ikev2 internal**, **debug radius authentication** logs capture EAP message exchanges between the IKEv2 client and EAP server through FlexVPN server. The relevant logs are highlighted.

- IKEv2 client skips AUTH payload in IKE_AUTH request to indicate intent to authenticate using EAP.

Click here to view code image

```
IKEv2:(SESSION ID = 69,SA ID = 1):Received Packet [From 172.16.1.1:59040/To
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST
Payload contents:
IKEv2-INTERNAL:Parse Vendor Specific Payload: (CUSTOM) VID IDi CERTREQ CFG SA
```

- The FlexVPN server sends AUTH payload in the IKE_AUTH response to authenticate itself. As the **query-identity** option is configured, the FlexVPN server also includes EAP-Request in the IKE_AUTH response to request the EAP identity.

Click here to view code image

```
IKEv2:(SESSION ID = 69,SA ID = 1):Generating EAP request
IKEv2:(SESSION ID = 69,SA ID = 1):Building packet for encryption.
Payload contents:
 VID IDr CERT CERT AUTH EAP
IKEv2:(SESSION ID = 69,SA ID = 1):Sending Packet [To 172.16.1.1:59040/From
172.16.1.2:4500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 1
IKEv2 IKE_AUTH Exchange RESPONSE
```

- IKEv2 client sends its EAP identity in EAP-Response message in the IKE_AUTH request message.

Click here to view code image

```
IKEv2:(SESSION ID = 69,SA ID = 1):Received Packet [From 172.16.1.1:59040/To
172.16.1.2:4500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 2
IKEv2 IKE_AUTH Exchange REQUEST
Payload contents:
 EAP
IKEv2:(SESSION ID = 69,SA ID = 1):Processing EAP response
IKEv2:Use authen method list TEST
IKEv2:pre-AAA: client sent User1 as EAP-Id response
```

- The FlexVPN server kicks off EAP authentication by sending the IKEv2 client's EAP identity to the EAP server as an EAP-Response in RADIUS Access-Request message.

Click here to view code image

```
IKEv2:sending User1 [EAP-Id] as username to AAA
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authentication request sent
RADIUS(00000049): Send Access-Request to 172.16.1.3:1812 id 1645/36, len 196
RADIUS:  authenticator 5A 04 CC CF DA 86 4C D3 - 4E B8 8A F3 75 47 52 81
RADIUS:  Service-Type          [6]   6   Login                      [1]
RADIUS:  Vendor, Cisco         [26]  26
RADIUS:   Cisco AVpair         [1]   20  "service-type=Login"
RADIUS:  Vendor, Cisco         [26]  28
RADIUS:   Cisco AVpair         [1]   22  "isakmp-phase1-id=router1"
RADIUS:  Calling-Station-Id    [31]  16  "172.16.1.2"
```

```
RADIUS:  Vendor, Cisco        [26]  63
RADIUS:   Cisco AVpair        [1]   57  "audit-session-
id=L2L40A683111ZO2L40A6869A2ZH1194E6AZO45"
RADIUS:  User-Name            [1]   4   "User1"
RADIUS:  EAP-Message          [79]  9
RADIUS:   02 3B 00 07 01 70 31                [ ;User1]
RADIUS:  Message-Authenticato[80]  18
RADIUS:   41 41 1E 2B 3B B9 4A 55 F0 1C 04 0A D4 60 62 86          [ AA+;JU`b]
RADIUS:  NAS-IP-Address       [4]   6   10.104.49.17
```

- The RADIUS server sends an EAP-Request in an Access-Challenge message, and the FlexVPN server relays it to the IKEv2 client in an IKE_AUTH response message.

Click here to view code image

```
RADIUS: Received from id 1645/36 172.16.1.3:1812, Access-Challenge, len 1159
RADIUS:  authenticator 52 73 15 7B 6F A2 46 38 - 7D 6B EB 3D 81 88 71 D0
RADIUS:  Service-Type         [6]   6   NAS Prompt                [7]
RADIUS:  EAP-Message          [79]  24

IKEv2:(SESSION ID = 69,SA ID = 1):Generating EAP request
IKEv2:(SESSION ID = 69,SA ID = 1):Building packet for encryption.
Payload contents:
 EAP
IKEv2:(SESSION ID = 69,SA ID = 1):Sending Packet [To 172.16.1.1:59040/From
172.16.1.2:4500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 2
IKEv2 IKE_AUTH Exchange RESPONSE
```

- IKEv2 client sends an EAP-Response message in the IKE_AUTH request message, and the FlexVPN server relays it to the RADIUS server in an Access-Request message.

Click here to view code image

```
IKEv2:(SESSION ID = 69,SA ID = 1):Received Packet [From 172.16.1.1:59040/To
172.16.1.2:4500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 3
IKEv2 IKE_AUTH Exchange REQUEST
Payload contents:
 EAP

RADIUS(00000049): Send Access-Request to 172.16.1.3:1812 id 1645/37, len 229
RADIUS:  authenticator 3B 58 59 F2 D3 AC F8 18 - 5C E8 6C C2 A9 BD 1B B9
RADIUS:  Service-Type         [6]   6   Login                     [1]
RADIUS:  Vendor, Cisco        [26]  26
RADIUS:   Cisco AVpair        [1]   20  "service-type=Login"
RADIUS:  Vendor, Cisco        [26]  28
RADIUS:   Cisco AVpair        [1]   22  "isakmp-phase1-id=router1"
RADIUS:  Calling-Station-Id   [31]  16  "172.16.1.2"
RADIUS:  Vendor, Cisco        [26]  63
RADIUS:   Cisco AVpair        [1]   57  "audit-session-
id=L2L40A683111ZO2L40A6869A2ZH1194E6AZO45"
RADIUS:  User-Name            [1]   4   "User1"
RADIUS:  EAP-Message          [79]  24
```

- The RADIUS server sends the EAP authentication status in an Access-Accept message, and the FlexVPN server relays it to the IKEv2 client in an IKE_AUTH response message.

Click here to view code image

```
RADIUS: Received from id 1645/37 172.16.1.3:1812, Access-Accept, len 1127
```

```
RADIUS:  authenticator 7B 7A B6 8B 64 E1 2D 03 - 86 74 B9 21 39 A7 1B FB
RADIUS:  Service-Type        [6]   6   NAS Prompt                [7]
RADIUS:  EAP-Message         [79]  6

IKEv2:(SESSION ID = 69,SA ID = 1):Sending EAP status message
IKEv2:(SESSION ID = 69,SA ID = 1):Building packet for encryption.
Payload contents:
 EAP
IKEv2:(SESSION ID = 69,SA ID = 1):Sending Packet [To 172.16.1.1:59040/From
172.16.1.2:4500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 3
IKEv2 IKE_AUTH Exchange RESPONSE
```

- If the EAP authentication was successful, the IKEv2 client sends the final IKE_AUTH request message to authenticate itself to the FlexVPN server using the pre-shared key method with the EAP shared secret or SK_pi and SK_pr values as the pre-shared key.

Click here to view code image

```
IKEv2:(SESSION ID = 69,SA ID = 1):Received Packet [From 172.16.1.1:59040/To
172.16.1.2:4500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 4
IKEv2 IKE_AUTH Exchange REQUEST
Payload contents:
 AUTH
```

- The FlexVPN server, after validating the AUTH payload from the IKEv2 client, performs the rest of the processing, including user and group authorization, processing of CFG_REQUEST, virtual-access interface creation, processing of IPsec Security Association proposal, construction of CFG_REPLY and AUTH payload to authenticate itself to the IKEv2 client using the pre-shared key method with the EAP shared secret or SK_pi and SK_pr values as the pre-shared key and sends the final IKE_AUTH response message.

Click here to view code image

```
IKEv2:(SESSION ID = 69,SA ID = 1):Building packet for encryption.
Payload contents:
 AUTH CFG SA TSi TSr NOTIFY(SET_WINDOW_SIZE) NOTIFY(ESP_TFC_NO_SUPPORT)
NOTIFY(NON_FIRST_FRAGS)
IKEv2:(SESSION ID = 69,SA ID = 1):Sending Packet [To 172.16.1.1:59040/From
172.16.1.2:4500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 4
IKEv2 IKE_AUTH Exchange RESPONSE
Payload contents:
 ENCR
```

- The **show crypto ikev2 sa detail** command displays the local and remote authentication methods and the peer's authenticated EAP identity.

Click here to view code image

```
Router#show crypto ikev2 sa detail
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         172.16.1.2/4500       172.16.1.2/59040      none/none             READY
      Encr: 3DES, PRF: SHA1, Hash: SHA96, DH Grp:2, Auth sign: RSA, Auth verify: EAP
      Life/Active Time: 900/21 sec
      CE id: 1028, Session-id: 22
      Status Description: Negotiation done
```

```
        Local spi: 123A08C78505C625       Remote spi: 3A1C6331D3A4726E
        Local id: 172.16.1.2
        Remote id: router1
        Remote EAP id: User1
```

The following XML configuration for AnyConnect can be used to create a profile for EAP-MSCHAPv2 to connect to a headend with the FQDN of vpn1.cisco.com. With regard to the headend certificate, if a subject alternative name (SAN) extension is present with relevant attributes, name verification is performed solely against the SAN. Relevant attributes include DNS name attributes for all certificates, and additionally include IP address attributes if the connection is being performed to an IP address. If a subject alternative name extension is not present, or is present but contains no relevant attributes, name verification is performed against any common name (CN) attributes found in the subject of the certificate. This identity must be configured as *vpn1.cisco.com*.

Click here to view code image

```
<ServerList>
  <HostEntry>
   <HostName>vpn1.cisco.com</HostName>
   <HostAddress>vpn1.cisco.com</HostAddress>
      <PrimaryProtocol>IPsec
      <StandardAuthenticationOnly>true
      <AuthMethodDuringIKENegotiation>EAP-MSCHAPv2</
AuthMethodDuringIKENegotiation>
      <IKEIdentity>cisco.com</IKEIdentity>
      </StandardAuthenticationOnly>
  </PrimaryProtocol>
  </HostEntry>
 </ServerList>
```

The following XML configuration for AnyConnect can be used to create a profile for EAP-MD5 to connect to a headend with the FQDN of vpn2.cisco.com. Note that the identity contained in the headend certificate must be vpn2.cisco.com.

Click here to view code image

```
<ServerList>
  <HostEntry>
   <HostName>vpn2.cisco.com</HostName>
   <HostAddress>vpn2.cisco.com</HostAddress>
      <PrimaryProtocol>IPsec
      <StandardAuthenticationOnly>true
      <AuthMethodDuringIKENegotiation>EAP-MD5</AuthMethodDuringIKENegotiation>
      <IKEIdentity>cisco.com</IKEIdentity>
      </StandardAuthenticationOnly>
  </PrimaryProtocol>
  </HostEntry>
 </ServerList>
```

AAA-based Pre-shared Keys

FlexVPN server supports hosting of symmetric and asymmetric pre-shared keys on an external AAA server, such as a RADIUS server for scalable management of pre-shared keys, which is useful when the pre-shared key method is used for local and/or remote authentication with a large number of IKEv2 initiators. The FlexVPN server uses AAA authorization to retrieve the pre-shared keys corresponding to a peer from the RADIUS server.

The pre-shared key lookup on the AAA server is based on the peer IKE identity. An IKEv2 name mangler can optionally be used to derive the AAA username from the peer IKE identity for pre-shared key lookup. Figure 9-7 illustrates retrieving pre-shared keys from the RADIUS server.

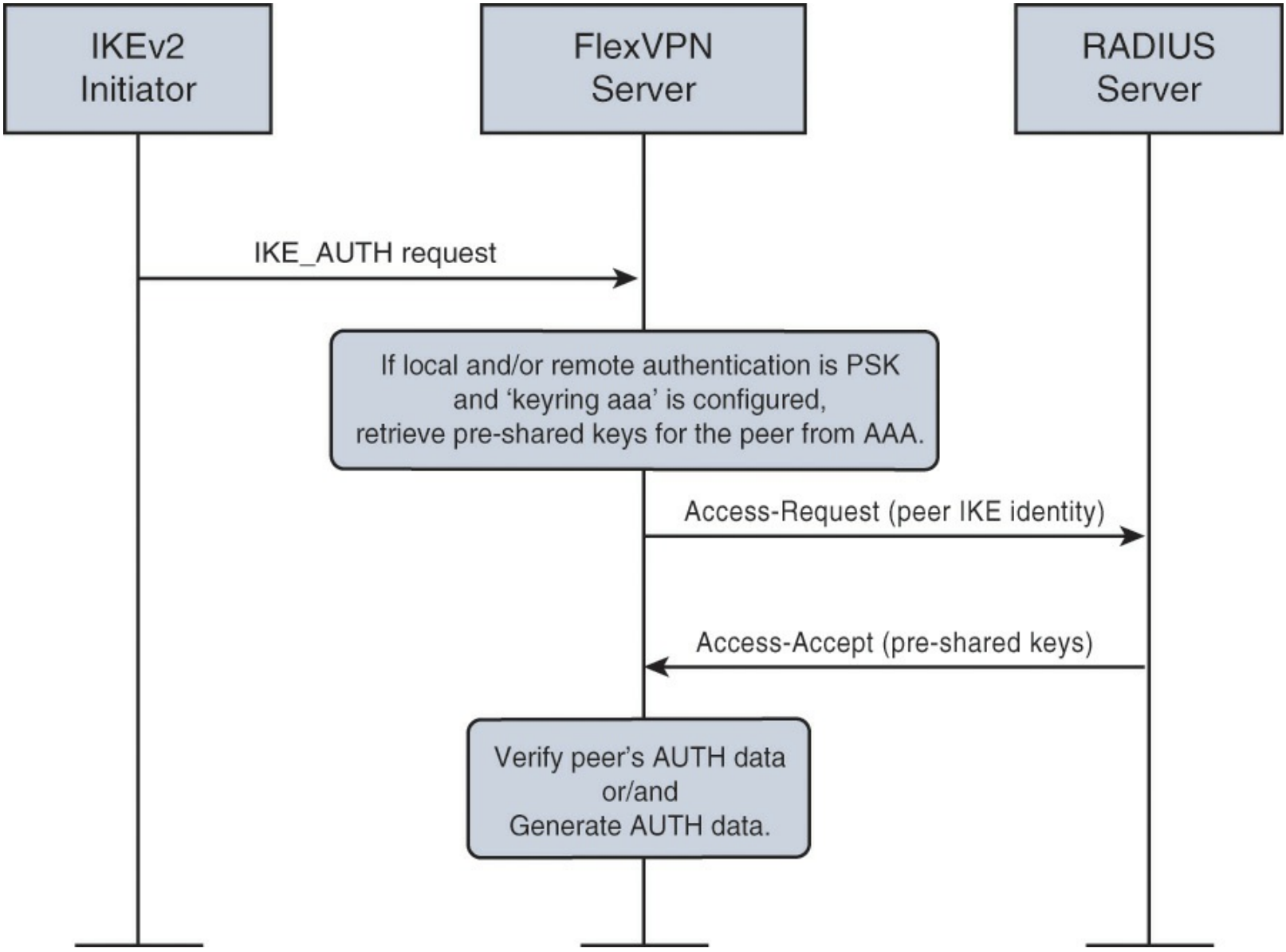


Figure 9-7 *Retrieving Pre-Shared Keys from the RADIUS Server*

Configuring AAA-based Pre-Shared Keys

Step 1. The AAA-based pre-shared key configuration is defined in the IKEv2 profile using the following command.

Click here to view code image

```
Router(config-ikev2-profile)#keyring ?
  aaa    AAA-based keyring
  local  Local keyring
```

Step 2. Specify the AAA authorization method list that specifies the AAA server hosting the keys.

Click here to view code image

```
Router(config-ikev2-profile)#keyring aaa ?
  WORD  AAA list name
```

Step 3. Specify an IKEv2 name mangler to derive the AAA username from the peer IKE identity

for the key lookup. If not specified, the entire peer IKE identity is used as AAA username for the key lookup.

Click here to view code image

```
Router(config-ikev2-profile)#keyring aaa aaa_psk_list ?
  name-mangler  Specify the name-mangler to derive username
  password      Specify the AAA password
  <cr>
```

RADIUS Attributes for AAA-Based Pre-Shared Keys

Following are the standard and the Cisco proprietary RADIUS attributes known as Cisco attribute-value(AV) pairs that are used to hold the symmetric and asymmetric pre-shared keys.

- The **Tunnel-Password** IETF attribute is used to hold the symmetric pre-shared key string.
- The **IKEv2-password-local** Cisco AV pair is used to hold the local pre-shared key that is used for local authentication, that is, used by the FlexVPN server to authenticate to the IKEv2 initiators (to generate the AUTH payload).
- The **IKEv2-password-remote** Cisco AV pair is used to hold the remote pre-shared key that is used for remote authentication, that is, used by the FlexVPN server to authenticate the IKEv2 initiators (to verify the peer's AUTH payload).
- AAA-based pre-shared keys allow for the ability to obtain additional attributes via RADIUS within the authentication phase to then be cached and used for implicit authorization. The ability to cache the attributes and reduce the RADIUS exchanges allows for an increase in the amount of tunnel setups that can occur at any given time.

AAA-Based Pre-Shared Keys Example

The following example illustrates sample configuration and logs on FlexVPN server for AAA-based pre-shared keys. In addition to pre-shared keys, any other policy attributes returned by the AAA server corresponding to the username used for pre-shared key lookup are cached and used as user authorization policy attributes when the **aaa authorization user** command with **cached** option is configured. This is known as implicit authorization. When the **cached** option is not configured, these policy parameters are ignored.

Step 1. Enable AAA.

```
aaa new-model
```

Step 2. Define the AAA authorization method list to specify the RADIUS server hosting the pre-shared keys.

Click here to view code image

```
aaa authorization network aaa_list1 group radius_group1
```

Step 3. Define the RADIUS server group.

Click here to view code image

```
aaa group server radius radius_group
 server name radius_server
```

Step 4. Define the RADIUS server.

Click here to view code image

```
radius server radius_server
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_key
```

Step 5. Define IKEv2 name mangler.

Click here to view code image

```
crypto ikev2 name-mangler name_mangler1
 fqdn hostname
```

Step 6. Configure AAA-based pre-shared keys in the IKEv2 profile.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share
 authentication remote pre-share
 keyring aaa aaa_list1 name-mangler name_mangler1
 aaa authorization user psk cached
```

The following **debug crypto ikev2**, **debug crypto ikev2 internal**, and **debug radius authentication** logs capture exchanges between the FlexVPN server and the RADIUS server to retrieve the pre-shared keys. The relevant debug outputs are highlighted.

- Note that the peer is authenticating using the pre-shared key method and is using an IKE identity of type FQDN with value ‘router1.domain1’.

Click here to view code image

```
IKEv2:(SESSION ID = 3,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 80753858A6D0BEAB - Responder SPI : EA3464A2957A6F65 Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST
IKEv2:(SESSION ID = 3,SA ID = 1):Searching policy based on peer's identity
'router1.domain1' of type 'FQDN'
IKEv2:(SESSION ID = 3,SA ID = 1):Peer's authentication method is 'PSK'
```

- FlexVPN server initiates an AAA authorization request to retrieve pre-shared keys from the AAA server.

Click here to view code image

```
IKEv2:(SESSION ID = 3,SA ID = 1):Get peer's preshared key for router1.domain1
IKEv2-INTERNAL:Fetching shared secret from AAA
IKEv2-INTERNAL:Name-mangler 'name_mangler1' returning mangled-name 'router1'
for name-type 1, name-len 15, name_string router1.domain1

IKEv2-INTERNAL: Using AAA method list aaa_list1 for peer router1
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Password request sent

AAA SRV(0000000E): process author req
AAA SRV(0000000E): Author method=SERVER_GROUP radius_group1
RADIUS/ENCODE: Best Local IP-Address 172.16.1.2 for Radius-Server 172.16.1.3
RADIUS(0000000E): Send Access-Request to 172.16.1.3:1645 id 1645/2, len 131
RADIUS:  authenticator 11 DD F2 16 3C 32 7A 9F - 26 6E AA DA 92 8D 99 8D
RADIUS:  User-Name           [1]   9   "router1"
```

```
RADIUS:  User-Password           [2]   18  *
RADIUS:  Calling-Station-Id      [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco           [26]  60
RADIUS:   Cisco AVpair           [1]   54  "audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZO3"
RADIUS:  Service-Type            [6]   6   Outbound                  [5]
RADIUS:  NAS-IP-Address          [4]   6   172.16.1.2
RADIUS(0000000E): Sending a IPv4 Radius Packet
RADIUS(0000000E): Started 5 sec timeout
```

- RADIUS server returns the corresponding pre-shared keys.

Click here to view code image

```
RADIUS: Received from id 1645/2 172.16.1.3:1645, Access-Accept, len 172
RADIUS:  authenticator B0 26 56 33 0F 1D 11 07 - A2 BE 8F 63 DD 71 DB B1
RADIUS:  Vendor, Cisco           [26]  33
RADIUS:   Cisco AVpair           [1]   27  "ipsec:route-set=interface"
RADIUS:  Vendor, Cisco           [26]  30
RADIUS:   Cisco AVpair           [1]   24  "ipsec:route-accept=any"
RADIUS:  Vendor, Cisco           [26]  45
RADIUS:   Cisco AVpair           [1]   39  "ipsec:route-set=prefix 192.168.1.1/24"
RADIUS:  Vendor, Cisco           [26]  44
RADIUS:   Cisco AVpair           [1]   38  "ipsec:tunnel-password=pre_shared_key"
RADIUS(0000000E): Received from id 1645/2
AAA SRV(0000000E): protocol reply PASS for Authorization
AAA SRV(0000000E): Return Authorization status=PASS
```

- Note that along with the pre-shared key, other policy attributes are also received from the AAA server.

Click here to view code image

```
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received password response
IKEv2-INTERNAL:Received symmetric password from radius server
IKEv2-INTERNAL:Received cached attributes:
route-set-interface: 1, route-accept any tag:1 distance:1,

IKEv2:(SESSION ID = 3,SA ID = 1):Verification of peer's authentication data PASSED
```

Accounting

The FlexVPN server leverages AAA accounting to report the IKE/IPsec session up and down events and the session statistics such as encrypt/decrypt packet and byte count. RADIUS accounting allows for a simple method to monitor the network. RADIUS accounting is mandatory when using RADIUS to allocate IP addresses. Accounting must be enabled to ensure that IP address allocation is indicated to the server, allowing for the return of IP addresses to the pool when no longer needed. The following example illustrates FlexVPN accounting configuration and logs.

Click here to view code image

```
Router(config-ikev2-profile)#aaa accounting ?
  anyconnect-eap  AAA list to use when IKEv2 remote auth method is AnyConnect
                  EAP
  cert            AAA list to use when IKEv2 remote auth method is certificate
                  based
  eap             AAA list to use when IKEv2 remote auth method is EAP
  psk             AAA list to use when IKEv2 remote auth method is PSK
```

The accounting method list specifies the AAA server that will store the accounting records.

Click here to view code image

```
Router(config-ikev2-profile)#aaa accounting psk ?
  WORD  AAA list name
```

Step 1. Configure the AAA accounting method list.

Click here to view code image

```
aaa accounting network acc_list
 action-type start-stop
 group radius_group1
```

Step 2. Enable accounting in the IKEv2 profile.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 aaa accounting psk acc_list
```

The selected output from **debug crypto ikev2**, **debug aaa accounting**, and **debug radius accounting** logs capture the accounting start and stop record being sent to the AAA server when the IKE/IPsec session comes up and goes down.

- The accounting start record is sent after sending the IKE_AUTH response and successful establishment of the IKE/IPsec session. The accounting start record includes the peer IP address, and IKE identity among other things.

Click here to view code image

```
IKEv2:(SESSION ID = 8,SA ID = 1):Sending Packet [To 172.16.1.1:500/From
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 0A40B2679CC0B5C2 - Responder SPI : 6833E924871B8C81 Message id: 1
IKEv2 IKE_AUTH Exchange RESPONSE
Payload contents:
 ENCR

IKEv2:(SESSION ID = 8,SA ID = 1):IKEV2 SA created; inserting SA into database. SA
lifetime timer (86400 sec) started
IKEv2:(SA ID = 1):[IPsec -> IKEv2] Creation of IPsec SA into IPsec database PASSED
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up
AAA/ACCT/EVENT/(00000011): CALL START
AAA/ACCT/EVENT/(00000011): IPSEC TNL UP
AAA/ACCT/IPSEC-TUNNEL(00000011): Queueing record is START

IKEv2:(SA ID = 1):[IKEv2 -> AAA] Accounting start request sent successfully
RADIUS(00000011): Send Accounting-Request to 172.16.1.3:1646 id 1646/9, len 256
RADIUS:  authenticator 6A 7A 8F EA 4E 10 E9 CE - 8C C9 EC A8 C4 70 C7 9F
RADIUS:  Acct-Session-Id     [44]  10  "00000007"
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  60
RADIUS:   Cisco AVpair       [1]   54  "audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZO8"
RADIUS:  Vendor, Cisco       [26]  40
RADIUS:   Cisco AVpair       [1]   34  "isakmp-phase1-id=router1.domain1"
RADIUS:  Vendor, Cisco       [26]  37
RADIUS:   Cisco AVpair       [1]   31  "isakmp-initator-ip=172.16.1.1"
RADIUS:  User-Name           [1]   17  "router1.domain1"
```

```
RADIUS:  Vendor, Cisco        [26]  36
RADIUS:   Cisco AVpair        [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Authentic       [45]  6   Local                     [2]
RADIUS:  Acct-Status-Type     [40]  6   Start                     [1]
RADIUS:  NAS-IP-Address       [4]   6   172.16.1.2
RADIUS:  Acct-Delay-Time      [41]  6   0
```

- The accounting stop record is sent after the IKE/IPsec session is deleted. The accounting stop record includes the peer IP address, IKE identity, and the session encrypt/decrypt packet and byte count among other things.

Click here to view code image

```
AAA/ACCT/EVENT/(00000011): IPSEC TNL DOWN
IKEv2:in_octets 13020, out_octets 17640
IKEv2:in_packets 105, out_packets 105
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Accounting stop request sent successfully
AAA/ACCT/EVENT/(00000011): CALL STOP
AAA/ACCT/IPSEC-TUNNEL(00000011): STOP

RADIUS(00000011): Send Accounting-Request to 172.16.1.3:1646
id 1646/10, len 324
RADIUS:  authenticator 6E 18 B4 4F E0 B5 C0 09 - 9F B9 FC 01 5A D4 69 6F
RADIUS:  Acct-Session-Id      [44]  10  "00000007"
RADIUS:  Calling-Station-Id   [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco        [26]  60
RADIUS:   Cisco AVpair        [1]   54  "audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZO8"
RADIUS:  Vendor, Cisco        [26]  40
RADIUS:   Cisco AVpair        [1]   34  "isakmp-phase1-id=router1.domain1"
RADIUS:  Vendor, Cisco        [26]  37
RADIUS:   Cisco AVpair        [1]   31  "isakmp-initator-ip=172.16.1.1"
RADIUS:  User-Name            [1]   17  "router1.domain1"
RADIUS:  Acct-Authentic       [45]  6   Local                     [2]
RADIUS:  Vendor, Cisco        [26]  36
RADIUS:   Cisco AVpair        [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Session-Time    [46]  6   34
RADIUS:  Acct-Input-Octets    [42]  6   13020
RADIUS:  Acct-Output-Octets   [43]  6   17640
RADIUS:  Acct-Input-Packets   [47]  6   105
RADIUS:  Acct-Output-Packets  [48]  6   105
RADIUS:  Acct-Terminate-Cause[49]  6   none                      [0]
RADIUS:  Vendor, Cisco        [26]  32
RADIUS:   Cisco AVpair        [1]   26  "disc-cause-ext=No Reason"
RADIUS:  Acct-Status-Type     [40]  6   Stop                      [2]
RADIUS:  NAS-IP-Address       [4]   6   172.16.1.2
RADIUS:  Acct-Delay-Time      [41]  6   0
```

Per-Session Interface

The FlexVPN server dynamically instantiates a point-to-point tunnel interface known as *virtual-access interface* for every incoming IKEv2/IPsec session. The FlexVPN server instantiates the virtual-access interface after successful peer authentication and user and group authorizations.

Figure 9-8 illustrates the sequence of events that lead to virtual-access creation. If the IKEv2 profile matched, based on the peer IKE identity, has a virtual-template configured in it, then a virtual-access interface is instantiated from that virtual-template for the peer/session.

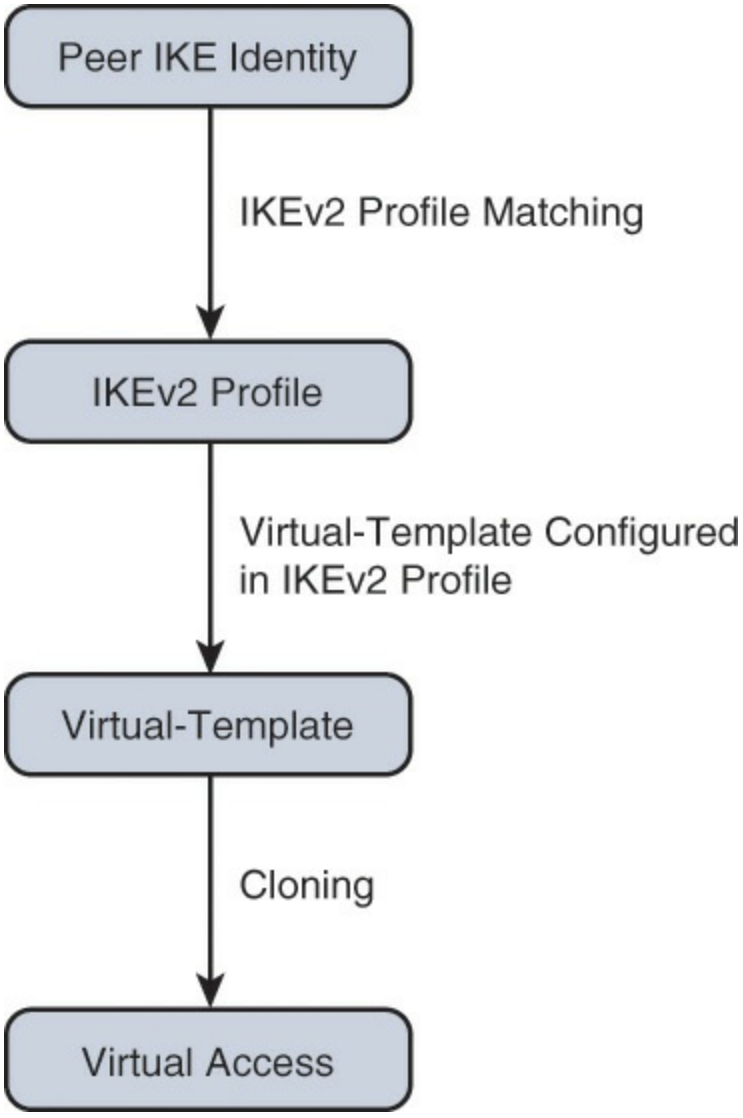


Figure 9-8 *Events Leading to Virtual-Access Creation*

The virtual-access interface derives its configuration from the following sources in the specified order.

1. Virtual-template interface
2. AAA authorization (optional)
3. Incoming IKE/IPsec session

Deriving Virtual-Access Configuration from a Virtual-Template

The FlexVPN server uses a virtual-template interface of type tunnel to instantiate the per session virtual-access interface. A virtual-template interface of type tunnel can be configured with all the commands that can be configured on a normal tunnel interface. The virtual-access interface inherits all the commands configured on the virtual-template it is cloned from.

The following are a few exceptions while configuring a virtual-template when compared to a normal point-to-point tunnel interface on the FlexVPN server

- The IP address on a virtual-template interface is configured using **ip unnumbered *interface*** and/or **ipv6 unnumbered *interface*** commands. These commands allow the virtual-template and the cloned virtual-access interfaces to borrow and reuse the IP address from the specified interface. This is required because Cisco IOS does not allow the same IP subnet to be

configured with **ip address** and **ipv6 address** commands on more than one interface due to the generation of erroneous routes and the loss of IP packets.

- The **tunnel destination** command is not configured on the virtual-template because the address of the remote peer is not known in advance. The cloned virtual-access interface derives the tunnel destination address from the source address of the incoming IKE/IPsec session.
- The **tunnel source** command may optionally be configured on a virtual-template, and when configured, it must match the source address of the incoming IKE/IPsec session. When not configured, the cloned virtual-access interface derives the tunnel source address from the destination address of the incoming IKE/IPsec session.

The following example illustrates virtual-template configuration on a FlexVPN server. The virtual-template interface is defined in the router global configuration mode and is referenced from the IKEv2 profile.

Click here to view code image

```
Router(config)#interface virtual-template ?
  <1-200>  Virtual-Template interface number

Router(config)#interface virtual-template 1 ?
  type  type of the virtual-template
  <cr>

Router(config)#interface virtual-template 1 type ?
  ethernet  Set VT type as ethernet
  serial    Set VT type as serial
  tunnel    Set VT type as tunnel
```

The virtual-template interface must be of type tunnel.

Click here to view code image

```
Router(config)#interface virtual-template 1 type tunnel ?
  <cr>
```

The IP address on the virtual-template must be configured using the **ip unnumbered** and/or **ipv6 unnumbered** command.

Click here to view code image

```
Router(config)#interface virtual-template 1 type tunnel
Router(config-if)#ip unnumbered e0/0
Router(config-if)#tunnel protection ipsec profile default
```

The virtual-template must be referenced from IKEv2 profile in order to be used for cloning virtual-access interfaces.

Click here to view code image

```
Router(config-ikev2-profile)#virtual-template 1
```

When a virtual-template of type tunnel has active cloned virtual-access interfaces, the interface configuration mode cannot be entered, and hence no new commands can be added to or any exiting commands deleted from or modified. A message is displayed as highlighted below.

Click here to view code image

```
Router#show vtemplate 1
Virtual-access subinterface creation is globally enabled

          Active     Active    Subint  Interface
        Interface Subinterface Capable Type
        --------- ------------ ------- ---------
Vt1             1            0    No   Tunnel

Router(config)#interface virtual-template 1 type tunnel
% Virtual-template config is locked, active vaccess present
Router(config)#
```

Note that when the virtual-template has no active virtual-access interfaces, the interface configuration mode can be entered, and commands can be added, deleted, or modified.

Click here to view code image

```
Router#show vtemplate 1
Virtual-access subinterface creation is globally enabled

          Active     Active    Subint  Interface
        Interface Subinterface Capable Type
        --------- ------------ ------- ---------
Vt1             0            0    No   Tunnel

Router(config)#interface virtual-template 1 type tunnel
Router(config-if)#
```

Deriving Virtual-Access Configuration from AAA Authorization

The virtual-access interface configuration can be hosted on an external AAA server and retrieved, using AAA authorization based on the peer IKE identity. This allows for scalable management of per-peer virtual-access configuration when the FlexVPN server needs to cater to a large number of IKEv2 initiators, with each requiring unique configuration such as VRF, access control list, QoS policy, firewall policy, IP MTU, and so on. A key benefit of AAA-based configuration is that it allows for cloning of virtual-access interfaces from a common virtual-template configured in a common IKEv2 profile for multiple peers while still being able to have per-peer configuration for each virtual-access interface. Without AAA-based configuration, per-peer configuration for each virtual-access requires a unique virtual-template configured in a unique IKEv2 profile per peer, which is not scalable.

The interface-config AAA Attribute

The AAA-based virtual-access configuration is accomplished using the Cisco proprietary **interface-config** AAA attribute of type string whose value can be applied as a command string on a virtual-access interface. The **interface-config** attribute is configured as a Cisco VSA (vendor-specific attribute) on a RADIUS server. For local AAA, the **interface-config** attribute can be configured, using an AAA attribute list referenced from the IKEv2 authorization policy. The **interface-config** attribute value is treated as an interface configuration mode command string and is applied to the virtual-access interface without any validation. Therefore, it must be ensured that the **interface-config** attribute is configured with a valid command string. Multiple **interface-config** attributes can be configured on the RADIUS server or the local AAA attribute list, one for each command.

The **interface-config** attribute values from the various authorization types are applied to the virtual-access interface in the following order.

1. Group authorization
2. Implicit authorization
3. User authorization

If the group authorization is configured to take precedence over user authorization, using the **aaa authorization group override** command, the **interface-config** attribute values are applied to the virtual-access interface in the following order.

1. Implicit authorization
2. User authorization
3. Group authorization

Note that in case of duplicate **interface-config** attributes, that is, the same interface command obtained from more than one authorization type, the one applied later will overwrite the previous. However, commands such as "**service-policy**" that require the existing command be removed before applying a new one, would not overwrite the previous command.

Deriving Virtual-Access Configuration from an Incoming Session

After the virtual-access interface is cloned from virtual-template and the **interface-config** attributes from AAA are applied, the following configuration is derived from the incoming IKE/IPsec session.

- The **tunnel destination** configuration is derived from the source address of the incoming IKE/IPsec session. Note that this is dynamic information and is not known in advance.
- The **tunnel source** configuration is derived from the destination address of the incoming IKE/IPsec session if the virtual-access interface is not already configured with the **tunnel source** command derived either from virtual-template or AAA.
- If the auto-detection of tunnel encapsulation protocol and the transport IP address family is enabled, using the **virtual-template** *number* **mode auto** in the IKEv2 profile, the tunnel mode configuration on the virtual-access interface is derived from the IPsec Security Association traffic selector protocol, and the IP address family used by the incoming IKE/IPsec session.

Virtual-Access Cloning Example

The following example illustrates the configuration and logs for virtual-access cloning from virtual-template and AAA authorizations. The example uses RADIUS server for user authorization and local AAA for group authorization.

Step 1. Define the AAA authorization method lists to specify the RADIUS server and the local AAA database hosting the per-peer virtual-access interface configuration.

Click here to view code image

```
aaa authorization network default local
aaa authorization network aaa_list1 group radius_group1
```

Step 2. Define the IKEv2 name mangler to derive AAA username for user and group authorizations.

Click here to view code image

```
crypto ikev2 name-mangler name_mangler1
```

```
 fqdn hostname

crypto ikev2 name-mangler name_mangler2
 fqdn domain
```

Step 3. Define the AAA attribute list and the IKEv2 authorization policy that is used with group authorization in this example. The AAA attribute list is referenced from the authorization policy. Note the interface-config attributes defined under the AAA attribute list as highlighted below.

Click here to view code image

```
aaa attribute list attr_list1
 attribute type interface-config "tunnel key 1"
 attribute type interface-config "ip mtu 1000"

crypto ikev2 authorization policy domain1
 aaa attribute list attr_list1
```

Step 4. Configure user and group authorizations in the IKEv2 profile.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 aaa authorization group psk list default name-mangler name_mangler2
 aaa authorization user psk list aaa_list1 name-mangler name_mangler1
 virtual-template 1
```

Step 5. Define the virtual-template interface of type tunnel. Note that the virtual-template is referenced from the IKEv2 profile.

Click here to view code image

```
interface Virtual-Template1 type tunnel
 ip unnumbered Ethernet0/0
 tunnel protection ipsec profile default
```

The selected output from **debug crypto ikev2**, **debug aaa per-user**, and **debug vtemplate cloning** logs captures the cloning of virtual-access interface from the virtual-template and the application of **interface-config** attributes from user and group authorizations to the virtual-access interface.

- Virtual-access interface is cloned from virtual-template1.

Click here to view code image

```
VT[Vi1]:Clone Vaccess from Virtual-Template1 (45 bytes)
VT[Vi1]:tunnel protection ipsec profile default
VT[Vi1]:end
```

- The **interface-config** attribute values from the group authorization are applied to the virtual-access interface.

Click here to view code image

```
INFO: AAA/AUTHOR: Processing PerUser AV interface-config
INFO: AAA/AUTHOR: Processing PerUser AV interface-config
VT[Vi1]:Clone Vaccess from AAA (30 bytes)
```

```
VT[Vi1]:tunnel key 1
VT[Vi1]:ip mtu 1000
VT[Vi1]:end
```

- The **interface-config** attribute values from the user authorization are applied to the virtual-access interface. Note that the same commands are received from group and user authorization. The commands from user authorization that are applied later overwrite the ones from group authorization applied earlier. Be aware of the order that commands are applied; these must be in the same order as commands are applied when manually configuring an interface. For example if an interface is to be placed into a VRF, using the **ip vrf forwarding** command, this must occur prior to the IP address of the interface being configured using the **ip address** command. If commands are not applied in the correct order, then one command can remove configuration applied by a previous command. Also, commands such as "**service-policy**" that require the existing command be removed before applying a new one would not overwrite the existing command.

Click here to view code image

```
INFO: AAA/AUTHOR: Processing PerUser AV interface-config
INFO: AAA/AUTHOR: Processing PerUser AV interface-config
VT[Vi1]:Clone Vaccess from AAA (30 bytes)
VT[Vi1]:ip mtu 1200
VT[Vi1]:tunnel key 2
VT[Vi1]:end
```

- The crypto session is associated with virtual-access1.

Click here to view code image

```
Router2#show crypto session
Crypto session current status
Interface: Virtual-Access1
Profile: default
Session status: UP-ACTIVE
Peer: 172.16.1.1 port 500
  Session ID: 28
  IKEv2 SA: local 172.16.1.2/500 remote 172.16.1.1/500 Active
  IPSEC FLOW: permit 47 host 172.16.1.2 host 172.16.1.1
        Active SAs: 2, origin: crypto-map
```

- The configuration on the virtual-access interface is derived from the virtual-template interface, AAA authorizations, and the IKE/IPsec session. Note that the tunnel destination and source are derived from the session.

Click here to view code image

```
Router2#show derived-config interface virtual-access 1
Building configuration...

Derived configuration : 222 bytes

interface Virtual-Access1
 ip unnumbered Ethernet0/0
 ip mtu 1200
 tunnel source 172.16.1.2
 tunnel destination 172.16.1.1
 tunnel key 2
 tunnel protection ipsec profile default
```

```
 no tunnel protection ipsec initiate
end
```

Auto Detection of Tunnel Transport and Encapsulation

The FlexVPN server supports automatic detection of tunnel encapsulation mode and transport IP protocol type in order to support initiators that use multiple encapsulation modes and transport IP types that use a single virtual-template interface and an IKEv2 profile. The auto-detect mode is enabled using the **virtual-template** *number* **mode auto** command in the IKEv2 profile that overrides the tunnel mode configuration derived from the virtual-template or AAA authorization. The cloned virtual-access interfaces instead derive the **tunnel mode** configuration from the IPsec Security Association traffic selector protocol and the IP address family used by the incoming IKE/IPsec session. The following example illustrates the configuration and logs for tunnel auto-mode. The **debug crypto IPsec** output and the show command outputs below illustrate that the tunnel encapsulation protocol and the transport IP protocol type and therefore, the **tunnel mode** command on the virtual-access interface is derived from the incoming IPsec Security Association proposal with protocol Encapsulation Security Payload overriding the configuration cloned from the virtual-template.

- IKEv2 profile with auto mode enabled.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 virtual-template 1 mode auto
```

- Virtual-template with encapsulation protocol GRE and transport IPv4:

Click here to view code image

```
Router#show interfaces virtual-template 1 | inc protocol/transport
  Tunnel protocol/transport GRE/IP
```

- IPsec proposal from peer with protocol as **any**:

Click here to view code image

```
IPSEC(validate_proposal_request): proposal part #1,
  (key eng. msg.) INBOUND local= 172.16.1.2:0, remote= 172.16.1.1:0,
    local_proxy= 0.0.0.0/0.0.0.0/256/0,
    remote_proxy= 0.0.0.0/0.0.0.0/256/0,
    protocol= ESP, transform= esp-aes esp-sha-hmac  (Tunnel),
```

- Virtual-access with encapsulation protocol Encapsulation Security Payload and transport IPv4:

Click here to view code image

```
Router#show interfaces virtual-access 1 | inc protocol/transport
  Tunnel protocol/transport IPSEC/IP
```

- Following is another example where the virtual-template is configured with **tunnel mode IPsec** while the peer proposes protocol GRE.

Click here to view code image

```
Router#show interfaces virtual-template 1 | inc protocol/transport
  Tunnel protocol/transport IPSEC/IP
```

- IPsec proposal from peer with protocol as GRE.

Click here to view code image

```
IPSEC(validate_proposal_request): proposal part #1,
  (key eng. msg.) INBOUND local= 172.16.1.2:0, remote= 172.16.1.1:0,
    local_proxy= 172.16.1.2/255.255.255.255/47/0,
    remote_proxy= 172.16.1.1/255.255.255.255/47/0,
    protocol= ESP, transform= esp-aes esp-sha-hmac  (Transport),

Router#show interfaces virtual-access 1 | inc protocol/transport
  Tunnel protocol/transport GRE/IP
```

RADIUS Packet of Disconnect

RFC5716, Dynamic Authorization Extensions to RADIUS, defines a way to delete the sessions and to change the authorization attributes associated with the sessions on the Network Access Server (NAS) initiated from the RADIUS server. The disconnect-request (also known as Packet-of-Disconnect, or PoD) message is used to delete a session and the change-of-authorization (CoA) message is used to change the authorization attributes associated with a session. The entity originating the unsolicited PoD/CoA request is known as the dynamic authorization client (DAC) and is typically co-resident with the RADIUS server. The NAS (FlexVPN server in this case) that processes the PoD/CoA messages is known as the dynamic authorization server (DAS). The PoD/CoA messages are sent to UDP port 3799 and are independent of the RADIUS Access-Request/Response, Access-Challenge, and Access-Accept/Reject messages.

The FlexVPN server supports the RADIUS packet of disconnect feature that allows deletion of an IKE/IPsec session through the unsolicited PoD message from the RADIUS server as illustrated in the Figure 9-9. The FlexVPN server currently supports only the **audit-session-id** Cisco AV pair in the PoD message to identify the session to be deleted. The FlexVPN server returns Disconnect-ACK after successfully deleting the specified session and returns Disconnect-NACK if session is not found.

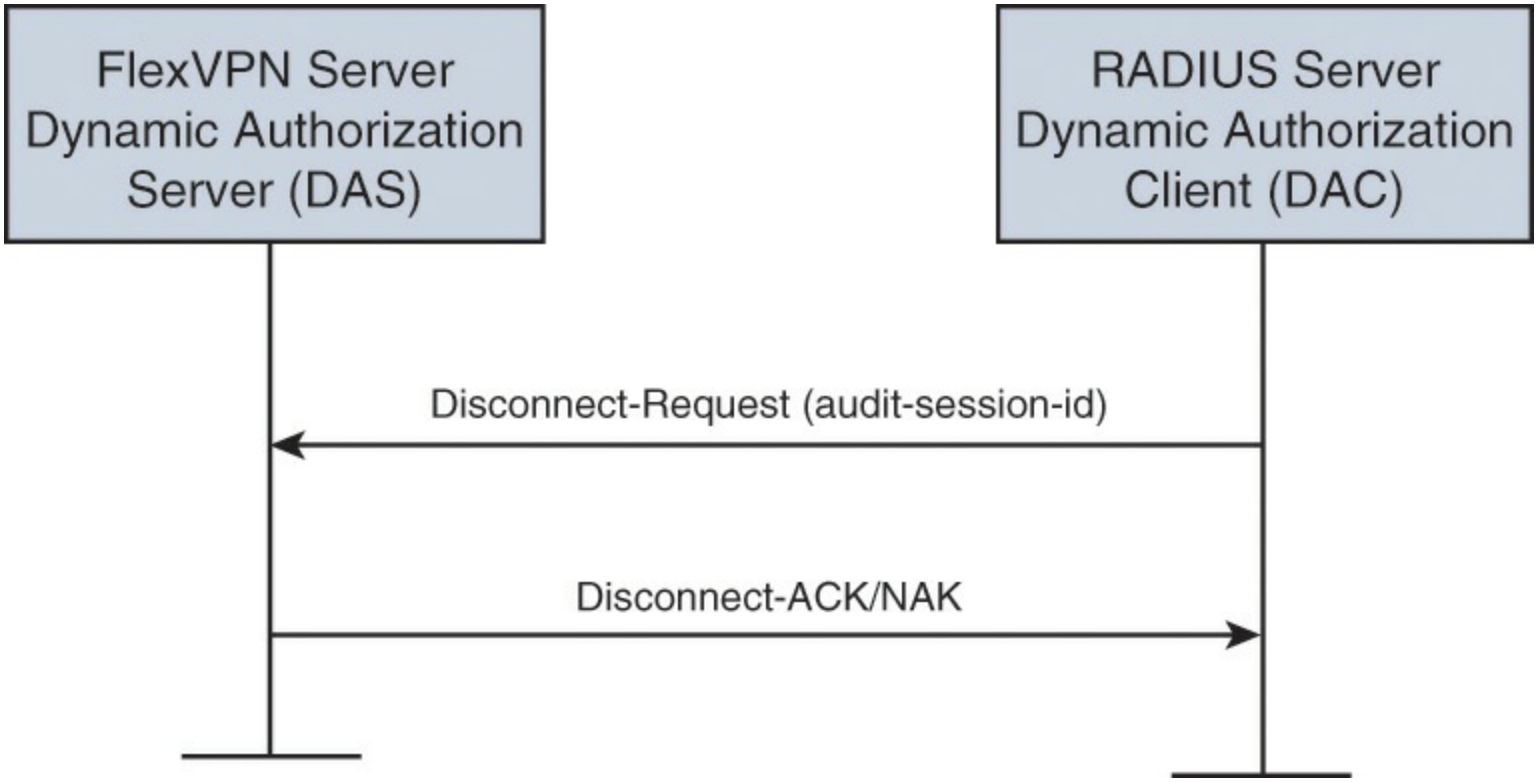


Figure 9-9 *FlexVPN Server Support for RADIUS PoD*

The **audit-session-id** to be used in the PoD message to identify the session to be deleted can be obtained from the debug and show command outputs on the FlexVPN server. The FlexVPN server sends the **audit-session-id** Cisco AV pair attribute in all the AAA authentication, authorization, and accounting requests and can be seen in the **debug radius authentication** and **debug radius accounting** output, and can also be seen using the **show aaa user** command.

Following are some of the use cases of PoD on the FlexVPN server:

- **Enforce re-authentication:** A network administrator might want to terminate a user session on a FlexVPN server to force a re-authentication if a session is connected for a very long duration.
- **Apply a new policy:** A network administrator might want to terminate an active crypto session and apply the new policy to the session when the client reconnects.
- **Free resources:** A session may need to be terminated to free the resources.

Configuring RADIUS Packet of Disconnect

To configure the RADIUS packet of disconnect follow the steps:

Step 1. Enable RADIUS dynamic authorization on the FlexVPN server.

Click here to view code image

```
Router(config)#aaa server radius ?
  dynamic-author  Local server profile for RFC 3576 support
Router(config)#aaa server radius dynamic-author
Router(config-locsvr-da-radius)#?
RADIUS Application commands:
  client      Specify a RADIUS client
  port        Specify port on which local radius server listens
  server-key  Encryption key shared with the radius clients
```

Step 2. Specify the dynamic authorization client (DAC) typically co-located with the RADIUS

server.

Click here to view code image

```
Router(config-locsvr-da-radius)#client ?
  Hostname or A.B.C.D  IP address of RADIUS client
  X:X:X:X::X           IPv6 address of RADIUS client
```

Step 3. Configure the shared-secret. Note that the same shared-secret must be configured on the DAC as well.

Click here to view code image

```
Router(config-locsvr-da-radius)#server-key ?
  0     Specifies an UNENCRYPTED key will follow
  6     Specifies an ENCRYPTED key will follow
  7     Specifies HIDDEN key will follow
  WORD  The UNENCRYPTED (cleartext) shared key
```

Step 4. On the DAC (RADIUS server), as part of the PoD command, specify the DAS (FlexVPN server) IP address and the audit-session-id identifying the session to be deleted.

RADIUS Packet of Disconnect Example

The following example illustrates the display of **audit-session-id** Cisco AV pair in AAA authorization and accounting requests and the deletion of IKE/IPsec session on receipt of RADIUS PoD. The relevant **debug aaa pod** and **debug crypto ikev2** logs are highlighted.

- The **audit-session-id** Cisco AV pair sent in authorization requests.

Click here to view code image

```
IKEv2:Using mlist aaa_list1 and username router1 for user author request
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorization request sent
RADIUS(000000D0): Send Access-Request to 172.16.1.3:1645 id 1645/37, len 132
RADIUS:  authenticator E0 93 9F 13 28 EB 8A EA - 64 46 2B 52 53 CF D1 33
RADIUS:  User-Name           [1]   9   "router1"
RADIUS:  User-Password       [2]   18  *
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  61
RADIUS:   Cisco AVpair       [1]   55  "audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN34"
```

- The 'audit-session-id' Cisco AV pair sent in accounting requests.

Click here to view code image

```
RADIUS(000000D3): Send Accounting-Request to 172.16.1.3:1646 id 1646/96, len 267
RADIUS:  authenticator DD 1D 68 F4 B0 EF 11 DA - 91 22 C2 9E 7C 86 5F A7
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  61
RADIUS:   Cisco AVpair       [1]   55  "audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN34"
```

- Display the **audit-session-id** in the **show aaa user** command output. Find the peer IKE identity from the **show crypto ikev2 sa detail** command.

Click here to view code image

```
Router2#show crypto ikev2 sa detail
 IPv4 Crypto IKEv2  SA
```

```
Tunnel-id Local                  Remote                 fvrf/ivrf              Status
1         172.16.1.2/500         172.16.1.1/500         none/none              READY
      Encr: AES-CBC, keysize: 256, PRF: SHA512, Hash: SHA512, DH Grp:5, Auth sign:
PSK, Auth verify: PSK
      Life/Active Time: 86400/3 sec
      CE id: 1057, Session-id: 52
      Status Description: Negotiation done
      Local spi: 8D331A95A67688AB       Remote spi: E7286487AC534733
      Local id: 172.16.1.2
      Remote id: router1.domain1
```

- Display the **audit-session-id** corresponding to the peer IKE identity, using the **show aaa user all** command. Note that AAA accounting must be enabled for the session for audit-session-id to be displayed.

Click here to view code image

```
Router#show aaa user all | begin router1.domain1
  IPSEC-TUNNEL: Username=router1.domain1
    Session Id=000000CE Unique Id=000000D8
    Start Sent=1 Stop Only=N
    stop_has_been_sent=N
    Method List=8655B740 : Name = acc_list
    Attribute list:
      86D5EEF8 0 00000082 formatted-clid(37) 10 172.16.1.1
      86D5EF2C 0 0000008A audit-session-id(819) 36
L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN35
      86D5EF60 0 00000081 isakmp-phase1-id(737) 15 router1.domain1
      86D5EF94 0 00000002 isakmp-initator-ip(738) 4 172.16.1.1
```

- The PoD received from the DAC (RADIUS server) with **audit-session-id** Cisco AV pair is shown:

Click here to view code image

```
RADIUS: POD  received from id 6 172.16.1.3:1700, POD Request, len 84
POD: 172.16.1.3 request queued
 ++++++ POD Attribute List ++++++
86D5EDB4 0 00000089 audit-session-id(819) 36
L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN34
```

- The FlexVPN server sends ACK after finding the session corresponding to the received **audit-session-id** and queueing it for deletion.

Click here to view code image

```
POD: Sending ACK from port 1700 to 172.16.1.3/1700
```

- The IKEv2/IPsec session is deleted.

Click here to view code image

```
IKEv2:(SESSION ID = 52,SA ID = 1):Sending DELETE INFO message for IKEv2 SA [ISPI:
0x5610ADC95CCD6AC7 RSPI: 0x1842E6A7EF7BE81A]
IKEv2:(SESSION ID = 52,SA ID = 1):Building packet for encryption.
Payload contents:
 DELETE NOTIFY(DELETE_REASON)
IKEv2:(SESSION ID = 52,SA ID = 1):Delete all IKE SAs
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Accounting stop request sent successfully
RADIUS(000000D3): Send Accounting-Request to 172.16.1.3:1646 id 1646/97, len 335
RADIUS:  authenticator 9E 05 29 DA EF 2B 44 37 - 89 5A DC F0 0B 71 70 FA
```

```
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  61
RADIUS:   Cisco AVpair       [1]   55  "audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN34"
```

- The FlexVPN server sends NACK when no matching session is found.

Click here to view code image

```
RADIUS: POD  received from id 9 172.16.1.3:1700, POD Request, len 84
POD: 172.16.1.3 request queued
 ++++++ POD Attribute List ++++++
86D5F0E4 0 00000089 audit-session-id(819) 36 L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN34

POD: 172.16.1.3 Unsupported attribute type 26 for component
POD: 172.16.1.3 user  0.0.0.0i sessid 0x0 key 0x0 DROPPED
POD: Added Reply Message: No Matching Session
POD: Added NACK Error Cause: Invalid Request
POD: Sending NAK from port 1700 to 172.16.1.3/1700
RADIUS:  18  21  4E6F204D61746368696E672053657373696F6E
RADIUS:  101 6   00000194
```

RADIUS Change of Authorization (CoA)

The FlexVPN server supports RADIUS change-of-authorization (CoA) as defined in RFC5716 that allows changing the authorization attributes applied to an IKE/IPsec session through the unsolicited CoA message from the RADIUS server as illustrated in the Figure 9-10. The FlexVPN server currently supports only the **audit-session-id** Cisco AV pair in the CoA message to identify the session. The FlexVPN server returns CoA-ACK after successfully applying the authorization attributes to the session, or it returns CoA-NACK if session is not found or the attributes could not be applied.

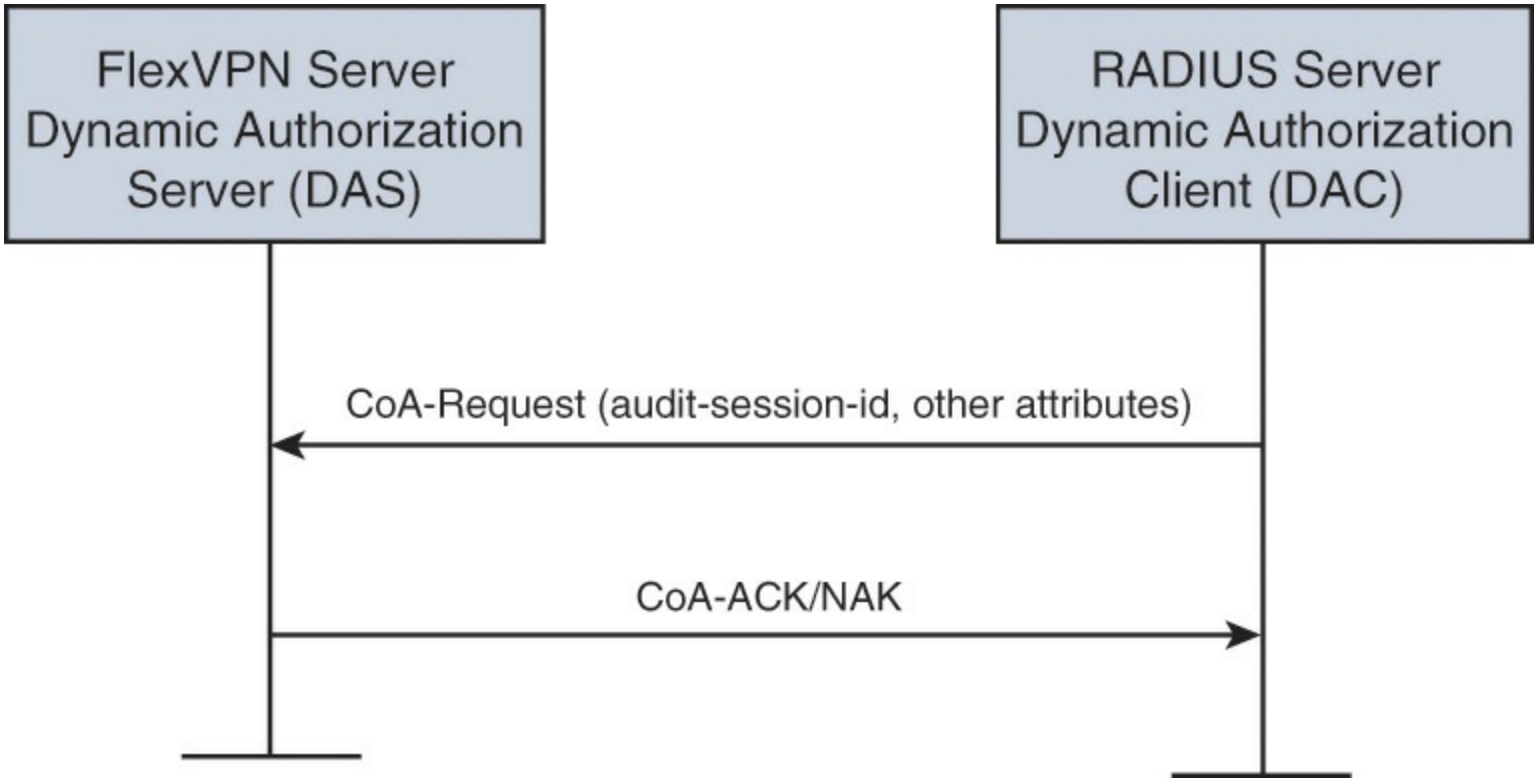


Figure 9-10 *FlexVPN Server Support for RADIUS CoA*

As with PoD, the **audit-session-id** to be used in the CoA message to identify the session can be obtained from the debug and show command outputs on the FlexVPN server. The FlexVPN server sends the **audit-session-id** Cisco AV pair attribute in all the AAA authentication, authorization, and

accounting requests and can be seen in the **debug radius authentication** and **debug radius accounting** output, as well as using the **show aaa user** command.

FlexVPN server currently supports the following RADIUS attributes in the CoA request:

- The **interface-config** Cisco AV pair attribute which can contain any interface configuration mode command string and is applied to the virtual-access interface for the session corresponding to the **audit-session-id** received in the CoA request.
- The **sub-policy-in** and **sub-qos-policy-in** Cisco AV pair attributes that contain the interface configuration mode command string to apply the QoS policy in the inbound direction (**service-policy input** *QoS-policy-name*). The command string is applied to the session virtual-access interface.
- The **sub-policy-out** and **sub-qos-policy-out** Cisco AV pair attributes that contain the interface configuration mode command string to apply the QoS policy in the outbound direction (**service-policy input** *QoS-policy-name*). The command string is applied to the session virtual-access interface.
- The **inacl** Cisco AV pair attribute, which defines a template access control list (ACL) that is applied to the session virtual-access interface in the inbound direction.
- The **outacl** Cisco AV pair attribute, which defines a template ACL that is applied to the session virtual-access interface in the outbound direction.

Configuring RADIUS CoA

To configure RADIUS CoA, follow these steps:

Step 1. Enable RADIUS dynamic authorization on the FlexVPN server.

Click here to view code image

```
Router(config)#aaa server radius ?
  dynamic-author  Local server profile for RFC 3576 support
Router(config)#aaa server radius dynamic-author
Router(config-locsvr-da-radius)#?
RADIUS Application commands:
  client      Specify a RADIUS client
  port        Specify port on which local radius server listens
  server-key  Encryption key shared with the radius clients
```

Step 2. Specify the dynamic authorization client (DAC) typically co-located with the RADIUS server.

Click here to view code image

```
Router(config-locsvr-da-radius)#client ?
  Hostname or A.B.C.D  IP address of RADIUS client
  X:X:X:X::X           IPv6 address of RADIUS client
```

Step 3. Specify the shared-secret that must be configured on the DAC, as well.

Click here to view code image

```
Router(config-locsvr-da-radius)#server-key ?
  0     Specifies an UNENCRYPTED key will follow
  6     Specifies an ENCRYPTED key will follow
  7     Specifies HIDDEN key will follow
```

```
  WORD  The UNENCRYPTED (cleartext) shared key
```

Step 4. On the DAC (RADIUS server), as part of the CoA command, specify the DAS (FlexVPN server) IP address, the audit-session-id identifying the session, and the attributes to be updated.

RADIUS CoA Examples

The following two examples illustrate the power of CoA to update the configuration on a virtual-access interface.

Updating Session QoS Policy, Using CoA

The following example illustrates updating the QoS policy applied to a session's virtual-access interface, using CoA. The QoS policies must be defined on the FlexVPN server.

Step 1. Define the QoS policies.

Click here to view code image

```
policy-map Input-QoS-policy-1
 class class-default

policy-map Input-QoS-policy-2
 class class-default

policy-map Output-QoS-policy-2
 class class-default

policy-map Output-QoS-policy-1
 class class-default
```

Step 2. Apply the QoS policies on the virtual-template so that they get cloned on the virtual-access interface.

Click here to view code image

```
interface Virtual-Template1 type tunnel
 ip unnumbered Ethernet0/0
 tunnel protection ipsec profile default
 service-policy input Input-QoS-policy-1
 service-policy output Output-QoS-policy-1
```

Step 3. Establish the session and verify the QoS policies applied on the virtual-access interface.

Click here to view code image

```
Router#show crypto session
Crypto session current status
Interface: Virtual-Access1
Profile: default
Session status: UP-ACTIVE
Peer: 172.16.1.1 port 500
  Session ID: 60
  IKEv2 SA: local 172.16.1.2/500 remote 172.16.1.1/500 Active
  IPSEC FLOW: permit 47 host 172.16.1.2 host 172.16.1.1
        Active SAs: 2, origin: crypto-map

Router#show derived-config interface virtual-access 1
interface Virtual-Access1
```

```
 ip unnumbered Ethernet0/0
 tunnel source 172.16.1.2
 tunnel destination 172.16.1.1
 tunnel protection ipsec profile default
 no tunnel protection ipsec initiate
 service-policy input Input-QoS-policy-1
 service-policy output Output-QoS-policy-1
end
```

Step 4. Note the **audit-session-id** for the session and use the same while initiating the CoA from the RADIUS server.

Click here to view code image

```
Router#show aaa user all | begin router1.domain1
  IPSEC-TUNNEL: Username=router1.domain1
    Session Id=000000F1 Unique Id=000000FB
    Start Sent=1 Stop Only=N
    stop_has_been_sent=N
    Method List=8655B740 : Name = acc_list
    Attribute list:
      86D5F338 0 00000082 formatted-clid(37) 10 172.16.1.1
      86D5F36C 0 0000008A audit-session-id(819) 36
      L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN3C
```

Step 5. Initiate the CoA command from the RADIUS server (DAC) to update the QoS policies applied on the session's virtual-access interface. The relevant **debug aaa coa** and **debug vtemplate cloning** logs are highlighted. Note that the audit-session-id and the attributes to update the QoS policies are received in the CoA request.

Click here to view code image

```
RADIUS: COA  received from id 1 172.16.1.3:1700, CoA Request, len 323
COA: 172.16.1.3 request queued
RADIUS:  authenticator 30 46 44 CE 19 6C FE 74 - 1D 64 7A DB 5B 9E 20
0F
RADIUS:  Vendor, Cisco       [26]  64
RADIUS:   Cisco AVpair       [1]   58  "ip:audit-session-id=
L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN3B"
RADIUS:  Vendor, Cisco       [26]  67
RADIUS:   Cisco AVpair       [1]   61  "ip:interface-config=
service-policy input Input-QoS-policy-2"
RADIUS:  Vendor, Cisco       [26]  69
RADIUS:   Cisco AVpair       [1]   63  "ip:interface-config=
service-policy output Output-QoS-policy-2"

 ++++++ CoA Attribute List ++++++
86D5F0E4 0 00000089 audit-session-id(819) 36
L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN3B
86D5F25C 0 00000089 interface-config(222) 39 service-policy input
Input-QoS-policy-2
86D5F290 0 00000089 interface-config(222) 41 service-policy output
Output-QoS-policy-2
```

The FlexVPN server sends CoA-ACK.

Click here to view code image

```
COA: Sending ACK from port 1700 to 172.16.1.3/1700
```

The interface configuration commands received in the **interface-config** attributes are

applied to the virtual-access interface for the session corresponding to the received audit-session-id.

Click here to view code image

```
VT[Vi1]:Clone Vaccess from AAA (99 bytes)
VT[Vi1]:service-policy input Input-QoS-policy-2
VT[Vi1]:service-policy output Output-QoS-policy-2
VT[Vi1]:end
```

Step 6. Verify that the QoS policies applied on the virtual-access interface are updated.

Click here to view code image

```
Router#show derived-config interface virtual-access 1
interface Virtual-Access1
 ip unnumbered Ethernet0/0
 tunnel source 172.16.1.2
 tunnel destination 172.16.1.1
 tunnel protection ipsec profile default
 no tunnel protection ipsec initiate
 service-policy input Input-QoS-policy-2
 service-policy output Output-QoS-policy-2
end
```

Updating the Session ACL, Using CoA

The following example illustrates applying a template ACL to a session's virtual-access interface using CoA. Note that the new ACL replaces any existing ACL applied to the virtual-access interface.

Step 1. Establish the session and verify the configuration on the virtual-access interface.

Click here to view code image

```
Router#show crypto session
Crypto session current status
Interface: Virtual-Access1
Profile: default
Session status: UP-ACTIVE
Peer: 172.16.1.1 port 500
  Session ID: 60
  IKEv2 SA: local 172.16.1.2/500 remote 172.16.1.1/500 Active
  IPSEC FLOW: permit 47 host 172.16.1.2 host 172.16.1.1
        Active SAs: 2, origin: crypto-map

Router#show derived-config interface virtual-access 1
interface Virtual-Access1
 ip unnumbered Ethernet0/0
 tunnel source 172.16.1.2
 tunnel destination 172.16.1.1
 tunnel protection ipsec profile default
 no tunnel protection ipsec initiate
end
```

Step 2. Note the **audit-session-id** for the session and use the same while initiating the CoA from the RADIUS server.

Click here to view code image

```
Router#show aaa user all | begin router1.domain1
  IPSEC-TUNNEL: Username=router1.domain1
    Session Id=000000F9 Unique Id=00000103
```

```
Start Sent=1 Stop Only=N
stop_has_been_sent=N
Method List=8655B740 : Name = acc_list
Attribute list:
  86D5EEF8 0 00000082 formatted-clid(37) 10 172.16.1.1

  86D5EF2C 0 0000008A audit-session-id(819) 36
  L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN3D
```

Step 3. Initiate the CoA command from the RADIUS server (DAC). The relevant **debug aaa coa** and **debug vtemplate cloning** logs are highlighted. Note that the audit-session-id and the attributes to apply to the template ACL are received in the CoA request.

Click here to view code image

```
RADIUS: COA  received from id 4 172.16.1.3:1700, CoA Request, len 151
COA: 172.16.1.3 request queued
RADIUS:  authenticator AD 54 06 14 D9 22 22 5C - 2E 06 C1 CF 86 56 41 C3
RADIUS:  Vendor, Cisco       [26]  33
RADIUS:   Cisco AVpair       [1]   27  "ip:inacl#3=permit 1.1.1.1"
RADIUS:  Vendor, Cisco       [26]  34
RADIUS:   Cisco AVpair       [1]   28  "ip:outacl#4=permit 1.1.1.3"
RADIUS:  Vendor, Cisco       [26]  64
RADIUS:   Cisco AVpair       [1]   58  "ip:audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN3D"

 ++++++ CoA Attribute List ++++++
86D5F0E4 3 00000081 inacl(144) 14 permit 1.1.1.1
86D5EEC4 4 00000081 outacl(310) 14 permit 1.1.1.3
86D5EEF8 0 00000089 audit-session-id(819) 36
L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN3D
```

The FlexVPN server sends CoA-ACK.

Click here to view code image

```
COA: Sending ACK from port 1700 to 172.16.1.3/1700
```

The commands are applied to the virtual-access interface for the session corresponding to the received audit-session-id.

Click here to view code image

```
VT[Vi1]:Clone Vaccess from AAA (86 bytes)
VT[Vi1]:IP access-group Virtual-Access1#2601 in
VT[Vi1]:IP access-group Virtual-Access1#2602 out
VT[Vi1]:end
```

Step 4. Verify that the ACLs on the virtual-access interface are updated.

Click here to view code image

```
Router#show ip interface virtual-access 1 | include access
  Outgoing access list is Virtual-Access1#2602, default is not set
  Inbound  access list is Virtual-Access1#2601, default is not set

Router#show ip access-lists
Standard IP access list Virtual-Access1#2601
    10 permit 1.1.1.1
Standard IP access list Virtual-Access1#2602
    10 permit 1.1.1.3
```

IKEv2 Auto-Reconnect

The FlexVPN server supports the IKEv2 auto-reconnect feature with AnyConnect client that allows the AnyConnect client to automatically reconnect the IKEv2 session without any user intervention when the IKEv2 session goes down temporarily due to loss of network connectivity, client switching to a different network, or going into sleep mode.

Following are some of the advantages of the auto-reconnect feature:

- The auto-reconnect IKE/IPsec session is much quicker and lighter.
  - For the auto-reconnect session, the AnyConnect client authenticates by using the pre-shared key method that is lighter and quicker than EAP and certificate-based methods. The session token provided by the FlexVPN server during the initial session is used as the pre-shared key.
  - The FlexVPN server preserves and reuses the AAA authorization attributes from the initial session for the auto-reconnect sessions, thus avoiding the overhead of round trips to the AAA servers.
- The reconnect feature enhances the user experience because it does not require any user intervention and automatically reconnects when the network connectivity is available or when the client wakes up from the sleep or hibernate mode.
- The reconnect feature enhances security because it ensures that the end host is always connected securely.

Limitations of the IKEv2 auto-reconnect feature are described below:

- The IKEv2 auto-reconnect feature is not supported with pre-shared key authentication. Hence, the AnyConnect client must authenticate, using either EAP or certificate-based methods, and the FlexVPN server client must authenticate using only the certificate-based method. Note that this limitation applies to the initial session and that the AnyConnect client authenticates using the pre-shared key method for the auto-reconnect sessions.
- The auto-reconnect does not work and the normal IKEv2 session establishment needs to happen in the following cases:
  - When the user disconnects the session on the AnyConnect client or clears the session on the FlexVPN server.
  - When the AnyConnect client host or the FlexVPN server reboots.
  - When the reconnect timer on the FlexVPN server expires after which the server discards the preserved session information.

Auto-Reconnect Configuration Attributes

The AnyConnect client indicates intent to use the auto-reconnect feature by requesting the Cisco proprietary configuration attributes in the CFG_REQUEST payload in IKE_AUTH request during the initial IKEv2 session setup. Note that there is no capability negotiation using the Vendor-ID payload for the auto-reconnect feature. If the auto-reconnect feature is enabled using **reconnect** command in the IKEv2 profile associated with the client session, the FlexVPN server provides values for the requested configuration attributes in the CFG_REPLY payload in IKE_AUTH response. Table 9-1 lists the Cisco proprietary configuration attributes related to the auto-reconnect feature.

Attribute	Description
reconnect-session-id	This attribute is used by the AnyConnect client as the IKE identity during the auto-reconnect sessions.
reconnect-token-id	This attribute is used by the AnyConnect client as the pre-shared key to authenticate to the FlexVPN server, using the pre-shared key method during the auto-reconnect sessions.
reconnect-dpd-interval	This attribute specifies the interval at which the AnyConnect client must send DPD keepalive probes. The FlexVPN server sets this value to 30 seconds.
reconnect-cleanup-interval	This attribute specifies the duration for which the FlexVPN server will preserve the session in inactive state on DPD failure detection after which it will delete the session. The client must reconnect before this duration. After this duration, the reconnect fails, and the client must establish a new session. This value is configured on the FlexVPN server using the **reconnect timeout** command, and the default value is 30 minutes.

Table 9-1 *Auto-Reconnect Attributes*

Note that the AnyConnect client uses the same **reconnect-session-id** and **reconnect-token-id** that was provided by the FlexVPN server during the initial session for all the reconnect sessions.

The auto-reconnect behavior for AnyConnect can be controlled through the AnyConnect XML profile with this setting:

Click here to view code image

```
<AutoReconnect UserControllable="true">true
  <AutoReconnectBehavior>ReconnectAfterResume</AutoReconnectBehavior>
</AutoReconnect>
```

Using this defined setting, AnyConnect will try to reconnect when the computer is brought back from sleep. The **AutoReconnectBehavior** preference defaults to **DisconnectOnSuspend.** For reconnection after resuming, the network administrator must either set **ReconnectAfterResume** in the profile or make the **AutoReconnect** and **AutoReconnectBehavior** preferences user controllable in the profile to allow users to set it.

Smart DPD

With the auto-reconnect feature, the FlexVPN server uses smart DPD probes to confirm if the AnyConnect client is unreachable before moving the session to the inactive state and waiting for the client to reconnect. The FlexVPN server automatically enables smart DPD probes when DPD is not explicitly configured and tracks the DPD probes from the client if available to check whether the client is live. When probes are not received from the client for a certain duration, the FlexVPN server starts sending probes to the client to confirm its liveness. If the client responds to the server probe, the server again waits for the probes from the client and repeats the process. If the client does not respond to server probes, the server concludes that the client is unreachable, preserves the session state, and moves the session to an inactive state, waiting for the client to reconnect. Note that when DPD is explicitly configured, the FlexVPN server uses these DPDs to track the client reachability.

Figure 9-11 illustrates the IKE_AUTH exchange for the initial and the auto-reconnect sessions.

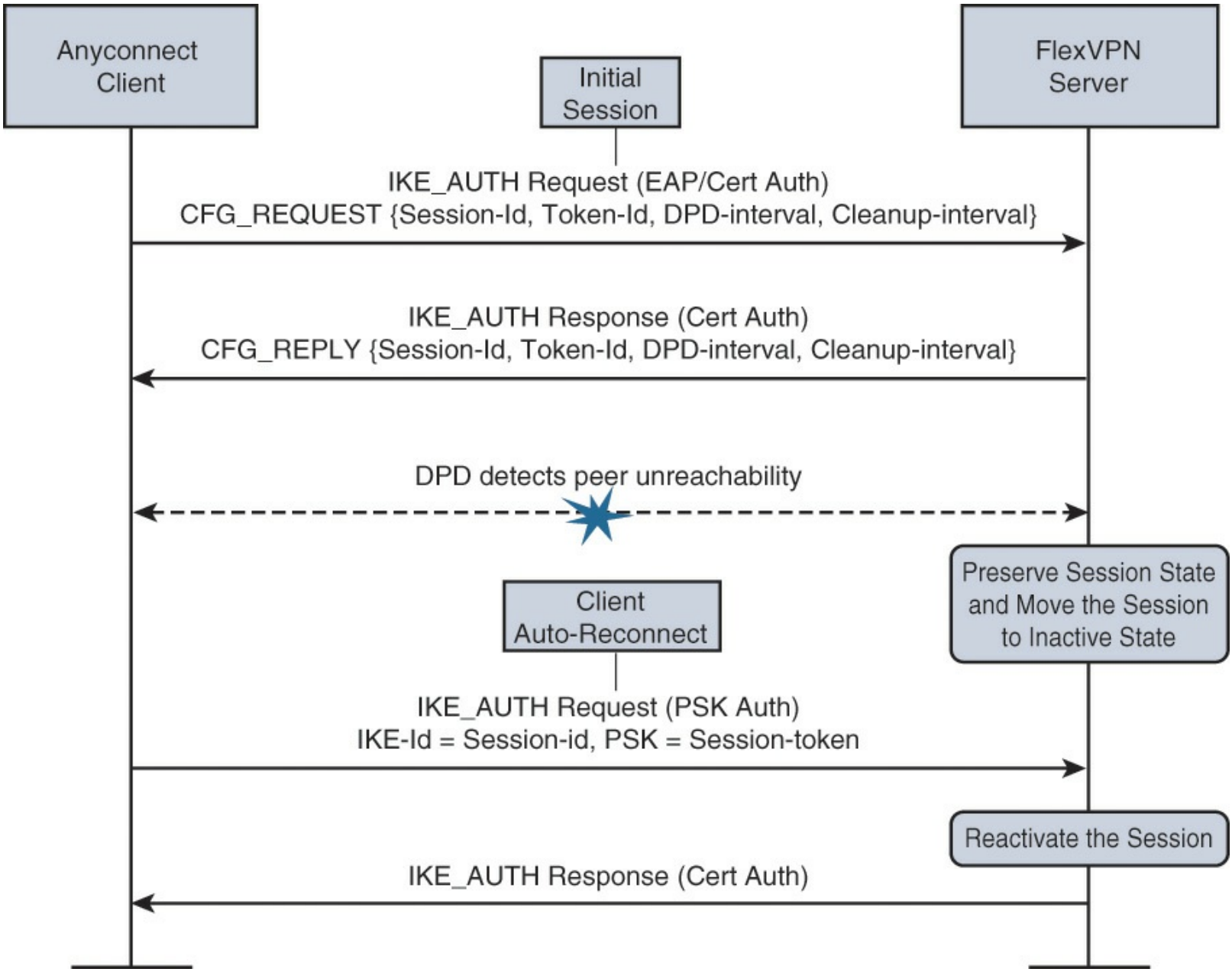


Figure 9-11 *IKEv2 Auto-Reconnect feature*

- When the IKEv2 auto-reconnect feature is enabled on the AnyConnect client, the client authenticates either using either EAP or certificate-based method and requests for the Cisco proprietary configuration attributes session-id, session-token, dpd-interval and cleanup-interval

in the CFG_REQUEST payload in IKE_AUTH request.

- If the IKEv2 auto-reconnect feature is enabled on the FlexVPN server, using the **reconnect** command in the IKEv2 profile, the server authenticates using the certificate-based method and provides the values for these reconnect attributes in the CFG_REPLY payload in IKE_AUTH response.
- When the server detects that the client is unreachable using smart DPD, the server preserves the session state, including the reconnect attributes and the AAA authorization attributes, and moves the session to inactive state, waiting for client to reconnect before the reconnect timeout duration.
- If the client session went down due any reason other than user-initiated disconnection, the client reconnects, authenticating using pre-shared-key method and using the session-id and session-token attributes received from the server as the IKE identity and pre-shared-key respectively.
- The server looks up the preserved session, using the session-id, and authenticates the client, using the corresponding session-token. If the authentication is successful, the server creates a new session, using the preserved authorization attributes, thus avoiding the overhead of round trips to the AAA server. The server authenticates using the certificate-based method.
- If the client does not reconnect within the reconnect timeout duration configured on the server, the preserved session state is discarded. After reconnect timeout, the client reconnect attempts will fail, and the client must establish a new session.

Configuring IKEv2 Auto-Reconnect

The IKEv2 auto-reconnect feature on the FlexVPN server is enabled in the IKEv2 profile as illustrated in the following example. The **reconnect-timeout** value specifies the duration for which the FlexVPN server would preserve the inactive session state and wait for the client to reconnect. After that, the server will discard the state, reconnect attempts from the client will fail, and the client will have to establish a new session. The default **reconnect-timeout** duration is 30 minutes.

Auto-reconnect cannot be configured in an IKEv2 profile when either the local or remote authentication method is based on a pre-shared key.

Click here to view code image

```
Router(config)#crypto ikev2 profile default
Router(config-ikev2-profile)#authentication local pre-share
Router(config-ikev2-profile)#reconnect ?
  timeout  timeout value for session in reconnect state
  <cr>
Router(config-ikev2-profile)#reconnect
Reconnect can not be configured if either Keyring, PSK  authentication or PSK
authorization enabled in profile.

Router(config-ikev2-profile)#authentication local rsa-sig
Router(config-ikev2-profile)#authentication remote eap
Router(config-ikev2-profile)#reconnect timeout ?
  <600-86400>  reconnect timeout interval in seconds
```

Please refer AnyConnect documentation for steps to enable the IKEv2 auto-reconnect feature on the client.

The **show crypto session detail** command outputs can be used to determine if a session is the initial

or the reconnected session. The capability flag **R** would be set for the reconnected session and not for the initial session. Also note that for a reconnected session, the **Auth verify** field in the **show crypto ikev2 sa detail** command output must be PSK.

- Initial session:

Click here to view code image

```
ASR-HUB1#show crypto session detail
Crypto session current status

Code: C - IKE Configuration mode, D - Dead Peer Detection
K - Keepalives, N - NAT-traversal, T - cTCP encapsulation
X - IKE Extended Authentication, F - IKE Fragmentation
R - IKE Auto Reconnect

Interface: Virtual-Access1
Profile: ike_profile
Uptime: 00:00:38
Session status: UP-ACTIVE
Peer: 10.104.105.162 port 64696 fvrf: (none) ivrf: (none)
      Phase1_id: eap
      Desc: (none)
  Session ID: 81
  IKEv2 SA: local 10.104.49.17/4500 remote 10.104.105.162/64696 Active
          Capabilities:DNX connid:1 lifetime:23:59:22

Router#show crypto ikev2 sa detail
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         10.104.49.17/4500     10.104.105.162/64696  none/none            READY
      Encr: 3DES, PRF: SHA1, Hash: SHA96, DH Grp:2, Auth sign: RSA, Auth verify: EAP
```

- Reconnected session:

Click here to view code image

```
Router#show crypto session detail
Crypto session current status

Code: C - IKE Configuration mode, D - Dead Peer Detection
K - Keepalives, N - NAT-traversal, T - cTCP encapsulation
X - IKE Extended Authentication, F - IKE Fragmentation
R - IKE Auto Reconnect

Interface: Virtual-Access1
Profile: ike_profile
Uptime: 00:02:49
Session status: UP-ACTIVE
Peer: 10.104.105.162 port 56028 fvrf: (none) ivrf: (none)
      Phase1_id: eap
      Desc: (none)
  Session ID: 84
  IKEv2 SA: local 10.104.49.17/4500 remote 10.104.105.162/56028 Active
          Capabilities:DNR connid:1 lifetime:23:57:11

Router#show crypto ikev2 sa detail
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
```

```
1          10.104.49.17/4500     10.104.105.162/56028  none/none            READY
      Encr: 3DES, PRF: SHA1, Hash: SHA96, DH Grp:2, Auth sign: RSA, Auth verify: PSK
```

The **show crypto ikev2 stats reconnect** displays the statistics relevant to the auto-reconnect feature. The following command output indicates that the FlexVPN server has received a total of 20 auto-reconnect requests from clients and that all of them succeeded—none failed. The **Reconnect capable active session count** message indicates the number of auto-reconnect capable sessions that have active sessions with the clients. The message **Reconnect capable inactive session count** indicates the number of auto-reconnect capable sessions that are inactive after detecting that client is unresponsive and have preserved the session state and are waiting for clients to reconnect before the reconnect timeout duration.

Click here to view code image

```
Router#show crypto ikev2 stats reconnect
Total incoming reconnect connection:        20
Success reconnect connection:               20
Failed reconnect connection:                0
Reconnect capable active session count:     5
Reconnect capable inactive session count:   2
```

User Authentication, Using AnyConnect-EAP

The AnyConnect client uses an XML-based aggregate authentication and configuration protocol that uses an XML document referred to as Aggregate XML to exchange authentication and configuration information between the client and the security gateway. IKEv2 can carry the Aggregate XML inside a Cisco proprietary configuration attribute or inside an EAP payload using a proprietary EAP method. Authentication performed using the Aggregate XML is referred to as Aggregate authentication.

AnyConnect-EAP

AnyConnect-EAP is a Cisco proprietary EAP method used between an AnyConnect client and security gateways such as ASA and the FlexVPN server for client-side user and device authentication using the XML-based aggregate authentication and configuration protocol. The FlexVPN server uses AnyConnect-EAP to query the client for user credentials and validates the credentials using AAA authentication similar to XAUTH user authentication in IKEv1. The AAA authentication can be local or RADIUS based. For local AAA authentication, the user credentials are configured using the **username** *name-string* **password** *password-string* commands in the global configuration mode. One advantage of the Anyconnect-EAP method is that it allows client user authentication without requiring an external EAP server. Unlike standards-based EAP methods (such as EAP-GTC, EAP-MD5 etc.), the FlexVPN Server does not operate in EAP pass-through mode. All EAP communication with the client terminates on the FlexVPN Server, and the required session key used to construct the AUTH payload is computed locally by the FlexVPN Server. The FlexVPN Server authenticates itself to the client using certificates as per RFC7296. Figure 9-12 illustrates user authentication with the AnyConnect-EAP method.

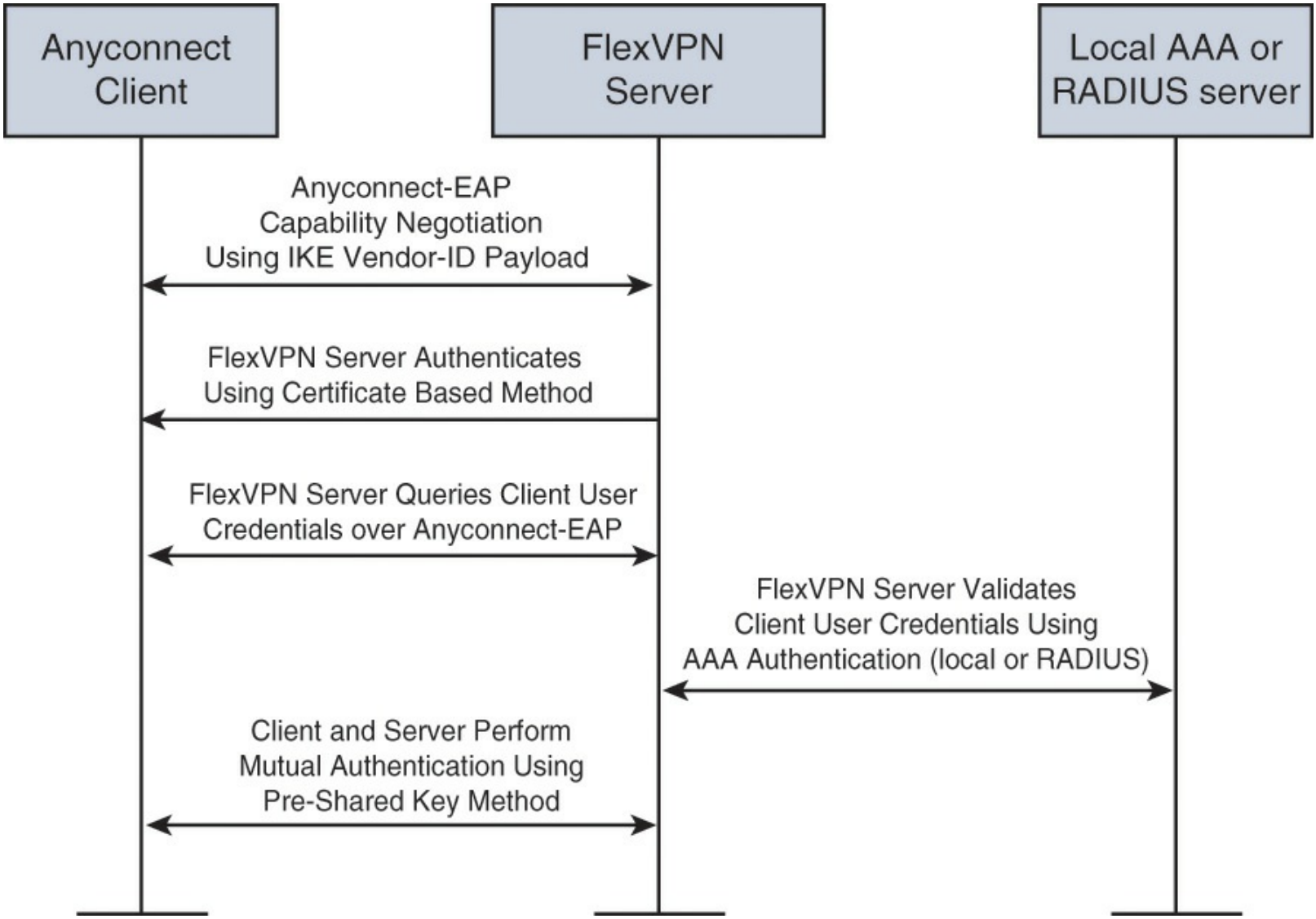


Figure 9-12 *User Authentication with AnyConnect-EAP*

AnyConnect-EAP XML Messages for User Authentication

Figure 9-13 illustrates the AnyConnect-EAP XML messages exchanged between the AnyConnect client and the FlexVPN server for client-side user authentication using RADIUS, however local AAA can also be used if required.

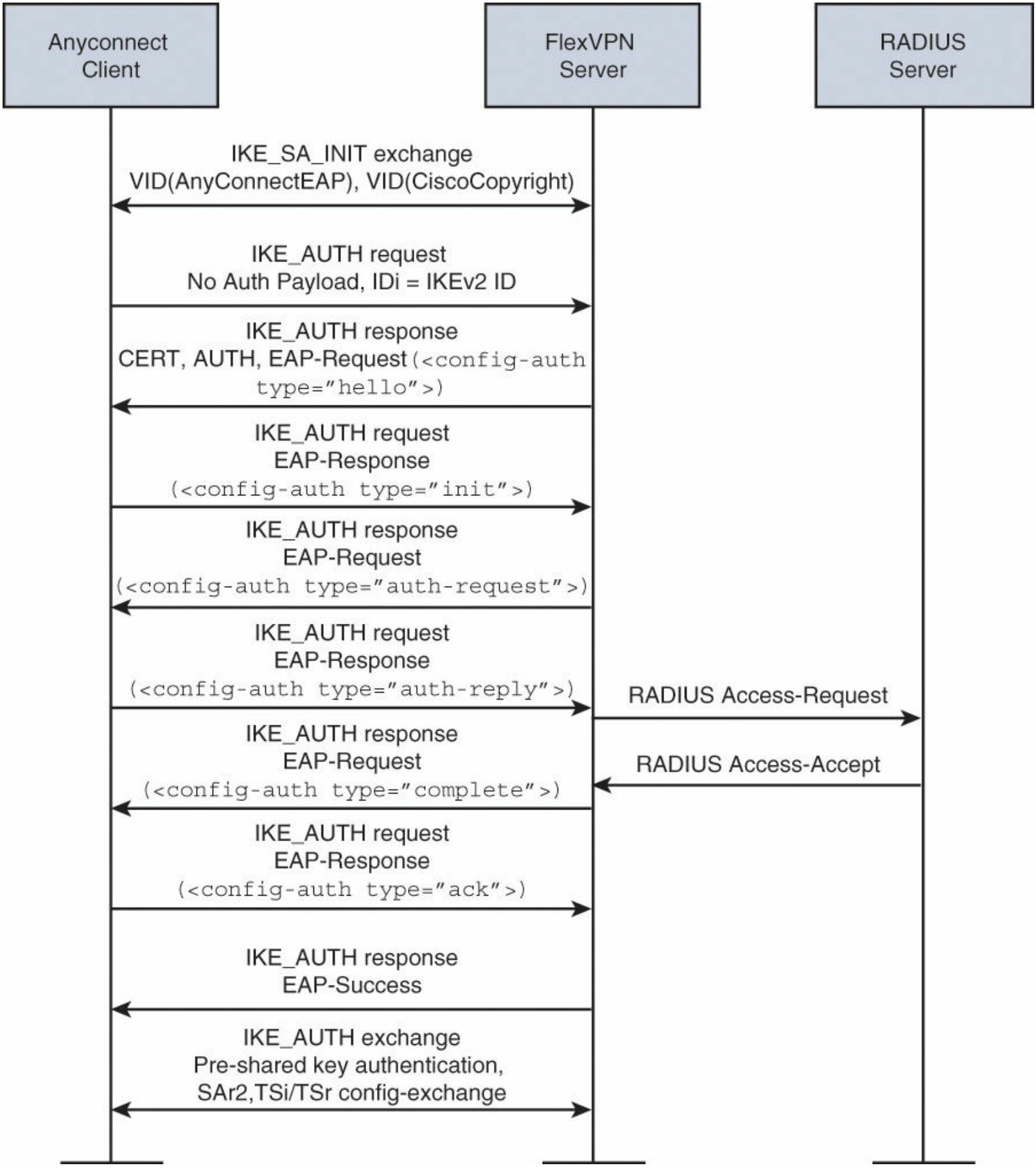


Figure 9-13 *AnyConnect-EAP XML Messages for User Authentication*

Following is a description of the steps involved:

1. The AnyConnect client and the FlexVPN server exchange the capability to perform AnyConnect-EAP using the **CiscoCopyright** and **AnyConnect-EAP** vendor ID payloads in IKE_SA_INIT exchange.

2. The AnyConnect client indicates its intent to authenticate using the AnyConnect-EAP method by leaving out the AUTH payload in the IKE_AUTH request. The IKE identity is defined within the XML profile on the AnyConnect client.
3. If the FlexVPN server is configured to authenticate this client, using the AnyConnect-EAP method in the IKEv2 profile using the **authentication remote anyconnect-eap** command, the FlexVPN server authenticates using certificate-based method and sends an EAP-Request containing AnyConnect-EAP XML **hello** message in IKE_AUTH response message. The **hello** message informs the client to initiate the aggregate authentication and configuration protocol with the client.
4. The AnyConnect client sends an EAP-Response containing AnyConnect-EAP XML **init** message in IKE_AUTH request message. The **init** message initiates the aggregate authentication and configuration protocol with the FlexVPN server.
5. The FlexVPN server sends an EAP-Request containing AnyConnect-EAP XML **auth-request** message in IKE_AUTH response message. The **auth-request** message requests authentication credentials from the client.
6. The AnyConnect client sends an EAP-Response containing AnyConnect-EAP XML **auth-reply** message in IKE_AUTH request message. The **auth-reply** message carries user credentials from the client.
7. The FlexVPN server validates the user credentials, using RADIUS authentication, and if the authentication is successful, sends an EAP-Request containing AnyConnect-EAP XML **complete** message in IKE_AUTH response message. The **complete** message indicates that the authentication is complete.
8. The AnyConnect client sends an EAP-Response containing AnyConnect-EAP XML **ack** message in IKE_AUTH request message. The **ack** message acknowledges the receipt of **complete** message.
9. The FlexVPN server sends an EAP-Success message in IKE_AUTH response message.
10. The AnyConnect client and the FlexVPN server perform mutual authentication using the pre-shared key method. The SK_pi and SK_pr values derived from the Diffie-Hellman shared secret are used as the pre-shared-key.

Configuring User Authentication, Using AnyConnect-EAP

The following example illustrates configuring AnyConnect-EAP on the FlexVPN server for the client user authentication using a RADIUS server, although local AAA can be used as well.

Step 1. AnyConnect-EAP can only be configured as a remote authentication method and the local authentication method must be certificate-based.

Click here to view code image

```
Router(config-ikev2-profile)#authentication remote ?
  anyconnect-eap  AnyConnect EAP
  eap             Extended Authentication Protocol
  ecdsa-sig       ECDSA Signature
  pre-share       Pre-Shared Key
  rsa-sig         Rivest-Shamir-Adleman Signature
Router(config-ikev2-profile)#authentication remote anyconnect-eap ?
```

```
  aggregate  use aggregate auth for anyconnect eap
Router(config-ikev2-profile)#authentication remote anyconnect-eap
aggregate
```

Step 2. Specify the AAA authentication list that points to the RADIUS server to authenticate the user credentials obtained through Anyconnet-EAP.

Click here to view code image

```
Router(config-ikev2-profile)#aaa authentication anyconnect-eap ?
  WORD  AAA list name
```

Step 3. The AAA user and group authorization and accounting can be configured for AnyConnect-EAP authentication method.

Click here to view code image

```
Router(config-ikev2-profile)#aaa authorization group ?
  anyconnect-eap  AAA list to use when IKEv2 remote auth method is
anyconnect
                  eap based

Router(config-ikev2-profile)#aaa authorization user ?
  anyconnect-eap  AAA list to use when IKEv2 remote auth method is
anyconnect
                  eap based

Router(config-ikev2-profile)#aaa accounting ?
  anyconnect-eap  AAA list to use when IKEv2 remote auth method is
AnyConnect
                  EAP
```

Step 4. The **show crypto ikev2 session detail** command output displays the remote authentication method as AnyConnect-EAP.

Click here to view code image

```
Router#show crypto ikev2 sa detail
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         192.168.1.1/4500      192.168.1.2/65058     none/none             READY
      Encr: 3DES, PRF: SHA1, Hash: SHA96, DH Grp:2, Auth sign: RSA,
Auth verify: AnyConnect-EAP
      Life/Active Time: 86400/8 sec
      CE id: 1007, Session-id: 6
      Status Description: Negotiation done
      Local spi: 016B20168FB2D051       Remote spi: B4E6576AD4BEAF7A
      Local id: 192.168.1.2
      Remote id: AnyConnect-IKE-ID
      Remote EAP id: user1@cisco.com
```

AnyConnect Configuration for Aggregate Authentication

When connecting using AnyConnect-EAP the client will present an IKE identity which is defined within the AnyConnect XML profile. The following illustrates the method of configuring the IKE Identity within the AnyConnect XML profile.

Click here to view code image

```
<IKEIdentity>AnyConnect-IKE-ID</IKEIdentity>
```

The following illustrates the method of configuring the FlexVPN server to match the IKE Identity. The same IKE Identity must be configured on the FlexVPN Server under the IKEv2 profile as what is presented by the AnyConnect client.

Click here to view code image

```
Router(config-ikev2-profile)#match identity remote ?
  address  IP Address(es)
  any      match any peer identity
  email    Fully qualified email string [Max. 255 char(s)]
  fqdn     Fully qualified domain name string [Max. 255 char(s)]
  key-id   key-id opaque string

Router(config-ikev2-profile)#match identity remote key-id ?
  WORD  Specify the key-id string

Router(config-ikev2-profile)#match identity remote key-id AnyConnect-IKE-ID
```

At the time of writing the feature that allows the AnyConnect client to download newer versions of the client software from the VPN gateway is not supported on Cisco IOS. In order to disable this feature a change in the AnyConnectLocalPolicy file is required as illustrated below.

<BypassDownloader>true</BypassDownloader>

Note: If configuring local authentication with AnyConnect-EAP the username and password must be defined locally on the IOS device using the **username** command.

Dual-factor Authentication, Using AnyConnect-EAP

IKEv2 supports EAP, pre-shared key, and certificate-based authentication methods. While pre-shared key and certificate-based methods are typically used for device authentication, EAP is mainly used for user authentication. However, since an IKEv2 node can authenticate using only one of these authentication methods, an IKEv2 node can either perform device or user authentication but not both. There are cases when using tunneled EAP methods (e.g., EAP-TLS) where multiple authentications can occur. However, this is not used for AnyConnect-EAP. The ability to perform device as well as user authentication by AnyConnect-EAP is referred to as dual-factor authentication here.

IKEv2 dual-factor authentication is useful and required for the following reasons.

- There are scenarios where making the authorization decision in IKEv2 requires using multiple authentication methods. For instance, it may be necessary to authenticate both the host machine requesting access and the user currently using the host. These two authentications would use two separate sets of credentials and might even use different authentication mechanisms.
- Another example is when an operator is hosting a virtual private network (VPN) gateway service for a third party, it may be necessary to authenticate the client to both the operator for billing purposes and the third-party's AAA server for authorizing access to the third-party's internal network.
- IKEv1 supports dual-factor authentication using XAUTH defined in the IETF draft "Extended Authentication within ISAKMP/Oakley (XAUTH)." The XAUTH user authentication is performed after device authentication, using pre-shared key or certificate-based methods. Customers migrating from IKEv1 expect IKEv2 to support both device and user authentication as well.

There are two ways of supporting dual-factor authentication.

- The RFC4739, "Multiple Authentication Exchanges in the Internet Key Exchange (IKEv2) Protocol," specifies an extension to IKEv2 that allows the use of multiple authentication exchanges, using either different mechanisms or the same mechanism. This extension allows, for instance, performing certificate-based authentication of the client host followed by an EAP authentication of the user.
- Double authentication using AnyConnect-EAP. The XML-based aggregate authentication and configuration protocol transported over the AnyConnect-EAP allows the security gateway to query the client for certificate for device authentication as well as user credentials for user authentication.

The FlexVPN server supports dual-factor authentication using the AnyConnect-EAP method as illustrated in the Figure 9-14.

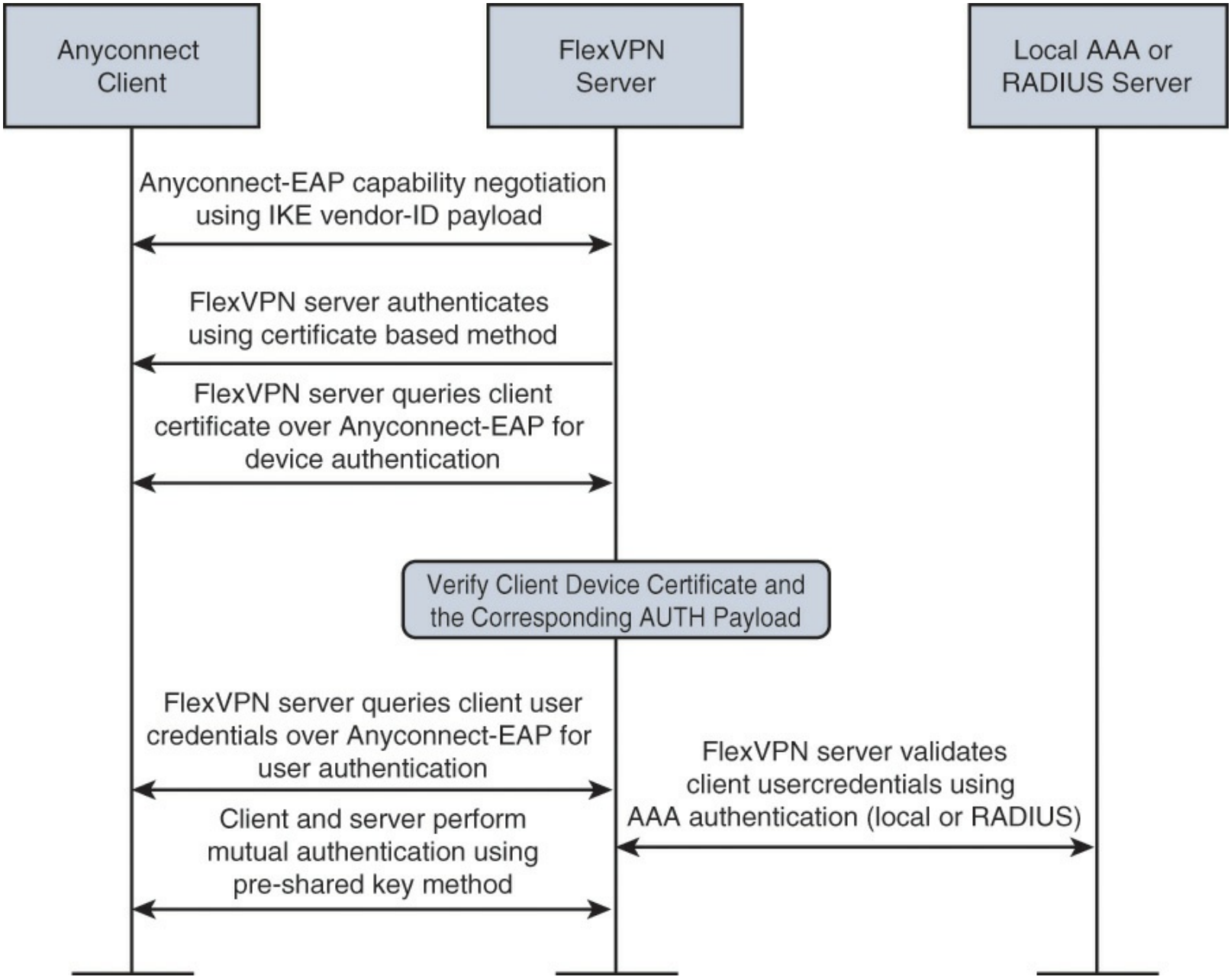


Figure 9-14 *Dual-Factor Authentication over AnyConnect-EAP*

AnyConnect-EAP XML Messages for dual-factor authentication

Figure 9-15 illustrates the AnyConnect-EAP XML messages exchanged between the AnyConnect client and the FlexVPN server for client-side device and user authentication. The AnyConnect-EAP message flow for dual-factor authentication is similar to that of user authentication except it requires an additional XML exchange for device authentication. Although the example illustrates AAA authentication using RADIUS server, local AAA can be used as well.

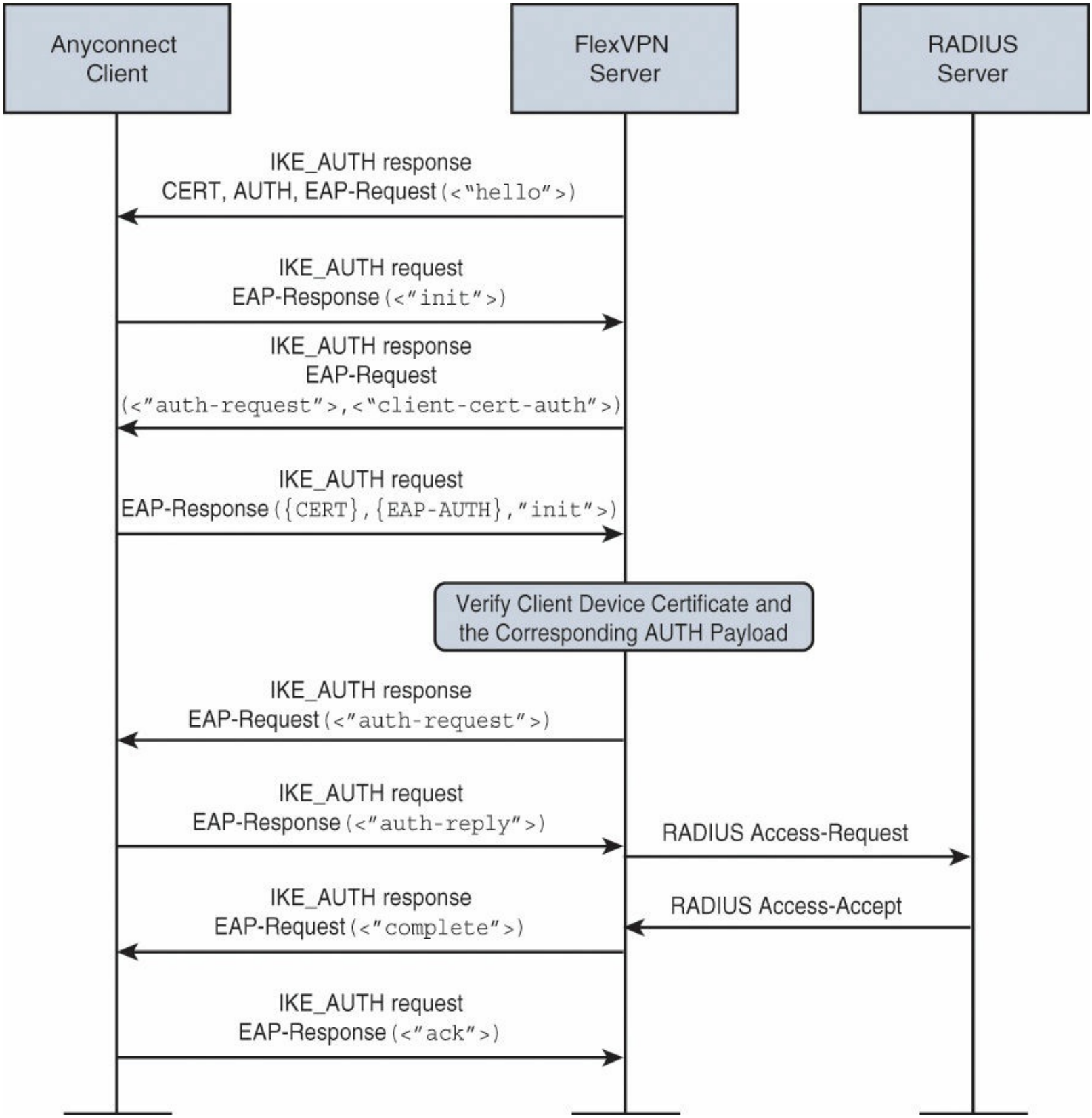


Figure 9-15 *AnyConnect-EAP XML Messages for Dual-Factor Authentication*

A description of the steps involved follows:

1. The AnyConnect client and the FlexVPN server exchange the capability to perform AnyConnect-EAP using the **CiscoCopyright** and **AnyConnect-EAP** vendor ID payloads in IKE_SA_INIT exchange.
2. The AnyConnect client indicates its intent to authenticate using the AnyConnect-EAP method by leaving out the AUTH payload in the IKE_AUTH request.
3. If the FlexVPN server is configured to authenticate this client using the AnyConnect-EAP method in the IKEv2 profile using the **authentication remote anyconnect-eap** command, the FlexVPN server authenticates with the certificate-based method and sends an EAP-Request containing AnyConnect-EAP XML **hello** message in IKE_AUTH response message. The **hello** message informs the client to initiate the aggregate authentication and configuration protocol with the client.
4. The AnyConnect client sends an EAP-Response containing AnyConnect-EAP XML **init** message in IKE_AUTH request message. The **init** message initiates the aggregate authentication and configuration protocol with the FlexVPN server.
5. The FlexVPN server sends an EAP-Request containing AnyConnect-EAP XML **auth-request** and **client-cert-auth** messages in IKE_AUTH response message, requesting the client certificate and the corresponding authentication data.
6. The AnyConnect client sends an EAP-Response containing a certificate and the corresponding authentication data signed using the private key and the **init** message in IKE_AUTH request message.
7. The FlexVPN server validates the certificate and the corresponding authentication data. If the authentication is successful, the FlexVPN server sends an EAP-Request containing the AnyConnect-EAP XML **auth-request** message in the IKE_AUTH response message. The **auth-request** message requests authentication credentials from the client.
8. The AnyConnect client sends an EAP-Response containing AnyConnect-EAP XML **auth-reply** message in IKE_AUTH request message. The **auth-reply** message carries user credentials from the client.
9. The FlexVPN server validates the user credentials using RADIUS authentication, and if the authentication is successful, sends an EAP request with AnyConnect-EAP XML **complete** message in IKE_AUTH response message. The **complete** message indicates that the authentication is complete.
10. The AnyConnect client sends an EAP-Response containing AnyConnect-EAP XML **ack** message in IKE_AUTH request message. The **ack** message acknowledges the receipt of **complete** message.
11. The FlexVPN server sends an EAP success message in IKE_AUTH response message.
12. The AnyConnect client and the FlexVPN server perform mutual authentication using the pre-shared key method. The SK_pi and SK_pr values derived from the Diffie-Hellman shared secret are used as the pre-shared key.

Configuring Dual-factor Authentication, Using AnyConnect-EAP

The following example illustrates configuring dual-factor authentication using AnyConnect-EAP on the FlexVPN server.

Click here to view code image

```
Router(config-ikev2-profile)#authentication remote anyconnect-eap aggregate ?
  cert-request  use double authentication during anyconnect eap
  <cr>

Router(config-ikev2-profile)#authentication remote anyconnect-eap aggregate
cert-request ?
  <cr>

Router2(config-ikev2-profile)#authentication remote anyconnect-eap aggregate
cert-request
```

Specify the AAA authentication list pointing to the RADIUS server to authenticate the user credentials obtained through Anyconnet-EAP.

Click here to view code image

```
Router(config-ikev2-profile)#aaa authentication anyconnect-eap ?
  WORD  AAA list name
```

RADIUS Attributes Supported by the FlexVPN Server

This section describes the RADIUS attributes supported by the FlexVPN server. Theattributes are either standard IETF RADIUS attributes or Cisco AV pairs and are configured on the RADIUS server or locally in the IKEv2 authorization policy that acts as the local AAA database. The attributes in the IKEv2 authorization policy are either configured as commands or by referencing an AAA attribute list. The FlexVPN server uses some of the attributes locally and pushes some to the client, using IKEv2 configuration payloads CFG_REPLY and CFG_SET. The attributes can be specific to a peer when obtained through user authorization and can apply to multiple peers when obtained through group authorization.

> **Note**
>
> Cisco AV pair is a Cisco vendor-specific attribute (VSA) with vendor-id 9 and vendor-type 1. The VSAs are encapsulated in the Radius IETF attribute 26 Vendor-Specific. The Cisco AV pair is specified as a string of format **protocol:attribute**=*value*.

The following are the RADIUS attributes that are used by the FlexVPN server locally.

- **Tunnel-password**=*key-string*

  This IETF attribute specifies the symmetric pre-shared key to be used with pre-shared key authentication.

- **ipsec:ikev2-password-local**=*key-string*

  This Cisco AV pair specifies the asymmetric local pre-shared key that is used by FlexVPN server to authenticate to peer(s).

- **ipsec:ikev2-password-remote**=*key-string*

This Cisco AV pair specifies the asymmetric remote pre-shared key that is used by FlexVPN server to authenticate peer(s).

- **Framed-Pool**=*pool-name*

  This IETF attribute specifies the name of the IPv4 address pool defined on the FlexVPN server that is used to allocate the IPv4 address to be assigned to the client using the INTERNAL_IP4_ADDRESS attribute. This attribute can be configured in the IKEv2 authorization policy using the **pool** *pool-name* command.

- **ipsec:group-dhcp-server**=*ipv4-address*

  This Cisco AV pair specifies the IPv4 DHCP server that is used by FlexVPN server to lease an IPv4 address to be assigned to the client using the INTERNAL_IP4_ADDRESS attribute. This attribute can be configured in the IKEv2 authorization policy using the **dhcp server** *ipv4-address | hostname command.*

- **ipsec:dhcp-giaddr**=*ipv4-address*

  This Cisco AV pair specifies the IPv4 address used by FlexVPN server as a source address to the contact the DHCP server. This attribute can be configured in the IKEv2 authorization policy using the **dhcp giaddr** *ipv4-address* command.

- **ipsec:dhcp-timeout**=*seconds*

  This Cisco AV pair specifies the time for which the FlexVPN server will wait for a response from the DHCP server. This attribute can be configured in the IKEv2 authorization policy using the **dhcp timeout** *seconds* command.

- **ipsec:ipv6-addr-pool**=*pool-name*

  This Cisco AV pair specifies the name of the IPv6 address pool defined on the FlexVPN server that is used to allocate the IPv6 address to be assigned to the client using the INTERNAL_IP6_ADDRESS attribute. This attribute is an IPv6 equivalent of the **Framed-Pool** IETF attribute. This attribute can be configured in the IKEv2 authorization policy using the **ipv6 pool** *pool-name* command.

- **ipsec:route-set=local ipv4** *address mask* **next-hop** *address* [**tag**:*tag*] [**distance:***distance*]

  This Cisco AV pair specifies an IPv4 subnet protected by the peer. The FlexVPN server adds a static route to this subnet pointing to the virtual-access interface corresponding to the peer and uses the specified next hop IP address, tag and distance while adding the static route. This attribute can be configured in the IKEv2 authorization policy using the **route set local ipv4** *address mask* **next-hop** *address* **tag** *tag* **distance** *distance* command.

- **ipsec:route-set=local ipv6** *prefix/length* **next-hop** *address* [**tag:***tag*] **[distance:***distance***]**

  This Cisco AV pair specifies an IPv6 subnet protected by the peer. The FlexVPN server adds a static route to this subnet pointing to the virtual-access interface corresponding to the peer and uses the specified next hop IP address, tag and distance while adding the static route. This attribute can be configured in the IKEv2 authorization policy using the **route set local ipv6** *prefix/length* **next-hop** *address* **tag** *tag* **distance** *distance* command.

- **ipsec:route-accept=any [tag**:*tag***] [distance:***distance***]**

This Cisco AV pair specifies the routes from the peer that can be accepted ("any" accepts all routes) and the tag and distance to be used while installing static routes to these subnets. This attribute can be configured in the IKEv2 authorization policy using the **route accept any [tag** *tag***] [distance** *distance***]** command.

- **ip:interface-config**=interface-command-string

  This Cisco AV pair specifies the interface configuration mode command string that is applied to the virtual-access interface associated with the peer's session.

- **ipsec:ipsec-flow-limit**=*limit*

  This Cisco AV pair specifies the maximum number of IPsec Security Associations an IKEv2 session can have. This attribute is used with dVTI multi-SA feature. This attribute can be configured in the IKEv2 authorization policy using the **IPsec flow-limit** *limit* command.

The following are the RADIUS attributes that are pushed by the FlexVPN server to the client using IKEv2 configuration payloads. These attributes are pushed to all the IKEv2 clients.

- **Framed-IP-Address**=*ipv4-address*

  This IETF attribute specifies the IPv4 address that is assigned to a remote access client using the INTERNAL_IP4_ADDRESS configuration attribute.

- **Framed-IP-Netmask**=*ipv4-subnet-mask*

  This IETF attribute specifies the IPv4 subnet mask associated with the IPv4 address assigned to the client. This attribute is pushed to client, using the INTERNAL_IP4_NETMASK configuration attribute. This attribute can be configured in the IKEv2 authorization policy, using the **netmask** *mask* command.

- **ipsec:route-set=interface** [*interface-name*]

  This Cisco AV pair enables the sending of IPv4/IPv6 address of the VPN interface (virtual-access interface associated with the peer's session) or the specified interface to the peer using the INTERNAL_IP4_SUBNET attribute. This attribute can be configured in the IKEv2 authorization policy using the **route set interface** [*interface-name*] command.

- **ipsec:route-set**=prefix *prefix/length*

  This Cisco AV pair specifies an IPv4 subnet protected by FlexVPN server and is pushed to the peer using the INTERNAL_IP4_SUBNET attribute.

- **ipsec:route-set=remote ipv4** *address mask*

  This Cisco AV pair specifies an IPv4 subnet protected by FlexVPN server and is pushed to the peer, using the INTERNAL_IP4_SUBNET attribute. This attribute can be configured in the IKEv2 authorization policy using the **route set remote ipv4** *address mask* command.

- **ipsec:route-set=remote ipv6** *prefix/length*

  This Cisco AV pair specifies an IPv6 subnet protected by FlexVPN server and is pushed to the peer using the INTERNAL_IP6_SUBNET attribute. This attribute can be configured in the IKEv2 authorization policy, using the **route set remote ipv6** *prefix/length* command.

- **ipsec:dns-servers**=*ipv4-address1*[*ipv4-address2*]

  This Cisco AV pair specifies the IPv4 addresses of the primary and the secondary DNS servers that are pushed to the client using the INTERNAL_IP4_DNS attributes. This attribute can be

configured in the IKEv2 authorization policy, using the **dns** *ipv4-address1*[*ipv4-address2*] command.

- **ipsec:wins-servers**=*ipv4-address1*[*ipv4-address2*]

  This Cisco AV pair specifies the IPv4 addresses of the primary and the secondary WINS servers that are pushed to the client using the INTERNAL_IP4_NBNS attributes. This attribute can be configured in the IKEv2 authorization policy, using the **wins** *ipv4-address1*[*ipv4-address2*] command.

- **ipsec:route-set=access-list** *acl-name | acl-number*

  This Cisco AV pair specifies an IPv4 access list defined on the FlexVPN server listing the IPv4 subnets protected by FlexVPN server that are pushed to the peer, using the INTERNAL_IP4_SUBNET attribute. This attribute can be configured in the IKEv2 authorization policy, using the **route set access-**list {*acl-name* | *acl-number*} command.

- **ipsec:route-set=access-list ipv6** *acl-name*

  This Cisco AV pair specifies an IPv6 access list defined on the FlexVPN server listing the IPv6 subnets protected by FlexVPN server that are pushed to the peer, using the INTERNAL_IP6_SUBNET attribute. This attribute can be configured in the IKEv2 authorization policy, using the **route set access-list ipv6** *acl-name* command.

- **ipsec:addrv6**=*ipv6-address*

  This Cisco AV pair specifies the IPv6 address that is assigned to a remote access client, using the INTERNAL_IP6_ADDRESS configuration attribute. This attribute is an IPv6 equivalent of the **Framed-IP-Address** IETF attribute.

- **ipsec:prefix-len**=*length*

  This Cisco AV pair specifies the prefix length of the IPv6 address assigned to the client. This attribute is pushed to client, using the INTERNAL_IP6_ADDRESS configuration attribute in the last (17th) byte. This attribute is an IPv6 equivalent of the **Framed-IP-Netmask** IETF attribute.

- **ipsec:ipv6-dns-servers-addr**=*ipv6-address1* [*ipv6-address2*]

  This Cisco AV pair specifies the IPv6 addresses of the primary and the secondary DNS servers that are pushed to the client, using the INTERNAL_IP6_DNS attribute. This attribute can be configured in the IKEv2 authorization policy, using the **ipv6 dns** *ipv6-address1*[*ipv6-address2*] command.

The following are the Cisco AV pair RADIUS attributes that are pushed by the FlexVPN server only to the Cisco clients such as FlexVPN and AnyConnect clients, using the Cisco proprietary unity configuration attributes. These attributes can be configured in the IKEv2 authorization policy as well.

- **ipsec:banner**=*banner-text*
- **ipsec:default-domain**=*domain-name*
- **ipsec:split-dns**=*name*
- **ipsec:ipsec-backup-gateway**=*name*
- **ipsec:pfs**=*value*
- **ipsec:include-local-lan**=*value*
- **ipsec:smartcard-removal-disconnect**=*value*

- **ipsec:configuration-url**=*url*
- **ipsec:configuration-version**=*version*

Remote Access Clients Supported by FlexVPN Server

FlexVPN server supports the following IKEv2 remote access clients:

- FlexVPN remote access client
- Cisco IKEv2 AnyConnect client
- Microsoft Windows7 IKEv2 Client
- Linux Strongswan IKEv2 client

FlexVPN Remote Access Client

The FlexVPN remote access client is a Cisco IOS based hardware client described in Chapter 10 "FlexVPN client."

Microsoft Windows7 IKEv2 Client

The Microsoft Windows 7 IKEv2 client sends an IP address as the Internet key exchange (IKE) identity that prevents the Cisco IKEv2 FlexVPN server from segregating remote users based on the IKE identity. To allow the Windows 7 IKEv2 client to send the email address (user@domain) as the IKE identity, apply the hotfix documented in KB975488 (http://support.microsoft.com/kb/975488) on Microsoft Windows 7 and specify the email address string in either the Username field when prompted or the CommonName field in the certificate depending on the authentication method.

For certificate-based authentication, the FlexVPN server and Microsoft Windows 7 client certificates must have an extended key usage (EKU) field as follows:

- For the client certificate, EKU field = client authentication certificate
- For the server certificate, EKU field = server authentication certificate

For EAP authentication, the Microsoft Windows 7 IKEv2 client expects an EAP identity request before any other EAP requests. Hence the **query-identity** option must be specified when configuring EAP as a remote authentication method in the IKEv2 profile, using **authentication remote eap** command, in order for the FlexVPN server to send an EAP identity request to the client.

Cisco IKEv2 AnyConnect Client

For certificate-based authentication, the FlexVPN server and the AnyConnect client certificates must have an extended key usage (EKU) field as follows:

- For the client certificate, EKU field = client authentication certificate
- For the server certificate, EKU field = server authentication certificate

If the FlexVPN server authenticates to an AnyConnect client using certificates, a **SubjectAltName** extension is required in the FlexVPN server certificate that contains the server's IP address or fully qualified domain name (FQDN). Additionally, HTTP certified URLs must be disabled on the FlexVPN server using the **no crypto ikev2 http-url cert** command.

The FlexVPN server supports the following features only with the AnyConnect client:

- IKEv2 auto-reconnect

- User authentication using AnyConnect-EAP
- Dual-factor authentication using AnyConnect-EAP

Summary

This chapter introduced the Cisco IOS FlexVPN server and then explained the various features supported by the FlexVPN server. The chapter started with an overview of the sequence of events on the FlexVPN server and then delved into the details of EAP authentication, AAA-based pre-shared keys and AAA accounting, and RADIUS PoD and CoA features. It explained the flow of messages between the client, server, and the RADIUS server. The chapter then explained the features that are specific to AnyConnect client, such as IKEv2 auto-reconnect and user and dual-factor authentication using the AnyConnect-EAP method. The chapter concluded with the RADIUS attributes and the clients supported by FlexVPN server.

Reference

https://tools.ietf.org/html/rfc2866 RADIUS Accounting.

Chapter 10. FlexVPN Client

Introduction

This chapter introduces and explains the features relevant to the FlexVPN client. The FlexVPN client is a Cisco IOS router-based VPN client positioned for remote offices and mobile workers to provide VPN connectivity to the corporate headquarters. The FlexVPN client leverages the IKEv2 configuration and the FlexVPN building blocks discussed in the earlier chapters. FlexVPN client uses a point-to-point tunnel interface with native IPsec or GRE encapsulation to connect to the FlexVPN server and supports EAP as a local authentication method along with pre-shared key and certificate-based authentication methods.

The chapter covers the following main topics:

- FlexVPN client overview
- Setting up the FlexVPN server
- EAP authentication
- Split-DNS
- WINS
- Domain name
- FlexVPN client profile
- Backup gateways
- Tunnel interface
- Tunnel initiation
- Dial backup
- Backup group
- Network address translation
- Design considerations
- Troubleshooting

FlexVPN Client Overview

The FlexVPN client is a hardware-based VPN client compared to other VPN clients such as Cisco AnyConnect, Microsoft Windows IKEv2 client, and Strongswan that are software based. While a software VPN client provides VPN access to a single host, the hardware-based FlexVPN client acts as a VPN gateway and provides VPN access for multiple hosts behind the client as illustrated in Figure 10-1.

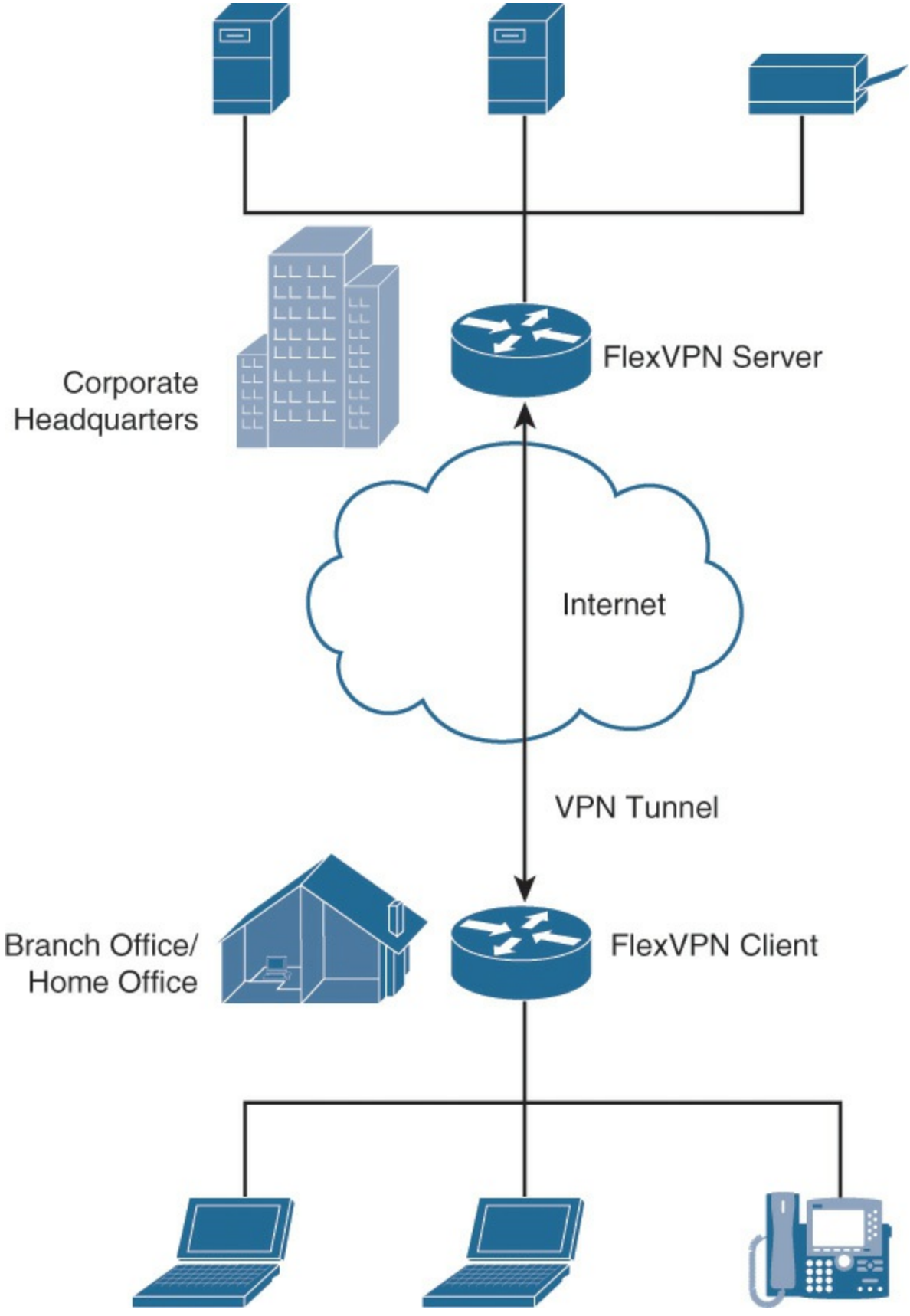


Figure 10-1 *FlexVPN Hardware-Based Client*

The FlexVPN client is positioned for remote offices and mobile workers, allowing for simplified deployments of a VPN, where the client could potentially connect to different VPN headends, depending on certain criteria such as route tracking, status of an SLA, or interface status. Services that are required for remote access solutions such as WINS, DNS, and DHCP and tightly integrated into the FlexVPN client and configuration can be pushed from the FlexVPN server using the IKEv2 configuration exchange that can integrate with DNS and DHCP. The client has a number of

similarities to the features included with EzVPN. However, IKEv2 is used instead of IKEv1 in EzVPN.

EzVPN is a legacy Cisco technology that was aimed at small office home office (SOHO) workers who required secure remote connectivity. The Cisco router-based EzVPN client creates an IPsec tunnel to the VPN headend and also provides services, such as DNS and DHCP, for devices connected to the router such as PCs and phones.

The FlexVPN client feature can be used with any type of FlexVPN deployment, including point-to-point, hub-spoke, and dynamic mesh topologies. Because the FlexVPN clients features integrates into any feasible deployment scenario, this now allows for deployments that would have previously required custom scripting or were simply impossible to implement.

The FlexVPN client feature allows a VPN to be established between a tunnel interface on the FlexVPN client and a FlexVPN server. The FlexVPN client profile is configured on a client and associated with a tunnel interface; the tunnel destination will be dynamically obtained from the client profile. The tunnel interface does not have a defined static destination, which allows for the enhanced object tracking feature to integrate with the FlexVPN client and allows the peer to be dynamically chosen depending on the status of track events.

Routing can be implemented either by static routing, a dynamic routing protocol, or by IKEv2 routing. Since a virtual-access interface is dynamically created on the FlexVPN server when a client connects, should a routing protocol be used, the routing peering must occur over this virtual-access interface.

The FlexVPN client functionality implements a number of features that are proprietary to Cisco, such as the attributes pushed from the FlexVPN server in the IKEv2 configuration exchange. An indication that a device is a FlexVPN client is the sending of Vendor Identification (VID) payloads containing CISCO-DELETE-REASON and FLEXVPN-SUPPORTED in the IKE_SA_INIT exchange.

FlexVPN Client Building Blocks

The FlexVPN client uses the following key building blocks:

- IKEv2 configuration exchange
- Static point-to-point tunnel interface
- FlexVPN client profile
- Object tracking
- NAT

IKEv2 Configuration Exchange

The FlexVPN client leverages the IKEv2 configuration exchange for the following:

- Request IPv4/IPv6 address from the server subnet and other parameters required for remote access, such as DNS and WINS from the FlexVPN server
- Learn the subnets protected by FlexVPN server in order to implement split tunneling
- Advertise the routable subnets behind the FlexVPN client to the FlexVPN server that the server can redistribute
- Exchange the tunnel address with the FlexVPN server in order to establish dynamic routing

protocols such as BGP over the tunnel

- Learn the FlexVPN servers that can used as backup

Static Point-to-Point Tunnel Interface

FlexVPN client uses a point-to-point tunnel interface to connect to the FlexVPN server. The tunnel interface is associated with a FlexVPN client profile that drives some of the tunnel configuration such as tunnel source and destination. The tunnel interface can use GRE or native IPsec encapsulation. GRE encapsulation enables dual-stack (IPv4 and IPv6 overlay) over IPv4 or IPv6 transport while the native IPsec encapsulation supports either IPv4 or IPv6 overlay (not both) over IPv4 or IPv6 transport, referred to as mixed mode. The tunnel IP address can either be statically configured or dynamically programmed with the IP address assigned by the FlexVPN server, which is enabled using the **ip address negotiated** command. The P2P tunnel interface allows applying features such as QoS, access control list, and firewall policy on the cleartext traffic in inbound and outbound directions. The features required on the encrypted traffic can be applied on the WAN interface.

FlexVPN Client Profile

The FlexVPN client profile is the central configuration construct that defines the peer list, tunnel initiation method, and advanced features, such as reactivate peer, backup gateways, and dial backup. The client profile references the tunnel interface and drives the dynamic configuration of tunnel source and destination based on the priority and availability of the WAN interfaces and the peers. The FlexVPN client profile is described in detail later in the chapter.

Object Tracking

The FlexVPN client leverages the Cisco IOS object-tracking infrastructure to support the various advanced features such as track-based tunnel activation, backup gateways, and dynamic tunnel source and destination configuration.

NAT

The network address translation (NAT) is applied on the FlexVPN client tunnel interface to translate the source address of the outbound cleartext traffic to the tunnel interface IP address, which is assigned by the FlexVPN server. This is useful when the hosts behind the FlexVPN client have private IP addresses that are not reachable beyond the FlexVPN client and hence are required to be translated to public IP address(es). The FlexVPN client can also apply NAT on the WAN interface for the Internet-bound, split tunnel traffic. Note that NAT must be explicitly configured on tunnel and outbound interfaces as needed.

FlexVPN Client Features

Following are some of the key features supported by FlexVPN client:

- Dual stack support
- EAP authentication
- IKEv2 routing and dynamic routing protocol support
- Support for EzVPN client and network extension modes
- Advanced features

Dual Stack Support

The FlexVPN client supports dual stack using GRE encapsulation. The FlexVPN client tunnel interface with GRE encapsulation mode (**tunnel mode gre**) can carry IPv4 and IPv6 overlay traffic over either IPv4 or IPv6 transport and with native IPsec encapsulation mode (**tunnel mode IPsec**) can carry either IPv4 or IPv6 overlay traffic over either IPv4 or IPv6 transport.

EAP Authentication

In addition to pre-shared keys and certificate-based authentication methods, the FlexVPN client can authenticate using the EAP-MSCHAPv2, EAP-GTC, and EAP-MD5 methods. Note that when the FlexVPN client authenticates using EAP, the FlexVPN server must authenticate using a certificate-based method.

Dynamic Routing

In addition to static routing and IKEv2 routing where the FlexVPN client and server exchange routes using an IKEv2 configuration exchange, the FlexVPN client supports dynamic routing protocol over the VPN tunnel. The FlexVPN client and server are configured to exchange the tunnel addresses, using an IKEv2 configuration exchange to establish dynamic routing on the overlay, that is, allow routing peering between the tunnel addresses. Note that dynamic routing is useful when the FlexVPN client has routable subnets and connections to a number of FlexVPN servers.

Support for EzVPN Client and Network Extension Modes

The FlexVPN client supports the EzVPN client mode by configuring NAT on the tunnel interface and translating the source addresses of the cleartext outbound traffic to the tunnel IP address that is assigned by the server. This is useful when the client subnets use private IP addresses.

The FlexVPN client supports the EzVPN network extension mode by advertising the routable subnets to the server, using IKEv2 routing or dynamic routing.

Advanced Features

The FlexVPN client supports various advanced features such as dynamic peer list, dynamic tunnel source, tunnel activation, dial backup, and peer reactivation features, using the Cisco IOS object-tracking infrastructure.

Setting up the FlexVPN Server

The FlexVPN server must be configured to accept connections from FlexVPN clients and send the required configuration attributes via the IKEv2 configuration exchange. The acceptance of client connections is configured by creating a virtual-template that is associated with an IKEv2 profile. Clients successfully connecting to the FlexVPN server will create a virtual-access interface from the virtual-template. The attributes sent to the client are configured either within an IKEv2 authorization policy or gained from a RADIUS server using AAA. These attributes are requested by the client in the IKE_AUTH request and passed to the client from the server in the IKE_AUTH response.

The virtual-template interface can be implemented, using either GRE encapsulation or native IPsec (**tunnel mode IPsec**). The tunnel mode on clients must match that on the FlexVPN server or the FlexVPN server must be configured with auto-detection of tunnel encapsulation protocol and

transport IP type as described in chapter 9 "FlexVPN Server."

In addition to the standard attributes, such as IP address/prefix, IPsec attributes, and routes, the FlexVPN server supports the following attributes that can be derived via AAA and are pushed to FlexVPN clients via the IKEv2 configuration exchange.

- Split-DNS
- Backup gateway addresses
- Default domain name
- Domain name system (DNS)
- Windows Internet name service (WINS)

Authentication of the FlexVPN client and server can be performed by EAP, pre-shared-keys, or certificates. If EAP is used, then only the client (being the initiator of the IKEv2 SA) can be authenticated by EAP and as per RFC7296, the server must be authenticated using certificates. An implementation using EAP MUST also use a public key-based authentication of the server to the client before the EAP authentication begins.

EAP Authentication

The FlexVPN client supports EAP as a local authentication method and supports the EAP-MSCHAPv2, EAP-GTC, and EAP-MD5 methods. EAP, as a local authentication method, is configured in the IKEv2 profile as illustrated in the following example.

Click here to view code image

```
FlexClient(config-ikev2-profile)#authentication local eap ?
  gtc       eap method gtc credentials
  md5       eap method md5 credentials
  mschapv2  eap method mschapv2 credentials
  <cr>
```

The EAP credentials can be statically configured in the IKEv2 profile.

Click here to view code image

```
FlexClient(config-ikev2-profile)#authentication local eap gtc ?
  password  EAP password
  username  EAP username

FlexClient(config-ikev2-profile)#authentication local eap md5 ?
  password  EAP password
  username  EAP username

FlexClient(config-ikev2-profile)#authentication local eap mschapv2 ?
  password  EAP password
  username  EAP username
```

If not configured in the IKEv2 profile, the user is prompted for EAP credentials during session activation. When the FlexVPN server is configured with a **query-identity** option, the server queries EAP identity, and the user is prompted for both username and password. When the **query-identity** option is not configured, the user is prompted only for the password. In this instance, the IKE identity is used as the EAP username. The following example illustrates the user prompt for EAP credentials on the FlexVPN client.

Following is the user prompt when the **query-identity** option is configured on the FlexVPN server.

Enter the command **crypto eap credential profile1**

Click here to view code image

```
  crypto eap credentials profile1
Enter the Username for profile profile1: user1
Enter the password for username cisco: password1
```

Note when using EAP, the username is treated as the IKE identity; in this case, if two devices have the same username and password combination they will not be able to connect to the same VPN headend at the same time. This is because the initial contact tears the first session down when the second session connects using the identical attributes.

Following is the user prompt when the **query-identity** option is not configured on the FlexVPN server.

Enter the command **crypto eap credentials profile1**

Click here to view code image

```
  crypto eap credentials profile1
Enter the password for username <IDi>: password1
```

Split-DNS

The Split-DNS function allows the FlexVPN client to act as a DNS proxy. The FlexVPN server pushes the domain names for split-DNS using the Cisco proprietary configuration attribute MODECFG_SPLITDNS_NAME.

In a typical scenario the FlexVPN client will have a DNS server configured that is on the service provider network and will act as a proxy. All DNS requests will be sent to this server from hosts on the internal protected network, even if the VPN is not active between the FlexVPN client and server. The split-DNS function enables a dynamic DNS view to be installed on the inside interface on creation of the VPN between the FlexVPN client and FlexVPN server. The inside interfaces are configured in the FlexVPN client profile, using the **client inside** *interface-name* command. When the inside interfaces are not configured in the client profile, the DNS view is applied to all interfaces except the tunnel interface and the tunnel source interfaces of all configured profiles. This DNS view will contain domain names and will then be used for requests from devices on the inside interface, which make requests for the specified domain names. The DNS view will take precedence over the default DNS server configured (that is, on the service provider network). When the FlexVPN client tunnel is torn down, the dynamic view is removed from the inside interface, and all DNS requests will be processed by the DNS server on the service provider network. Allowing for certain DNS requests to be processed by the service provider reduces the load on the corporate DNS server.

Figure 10-2 illustrates a scenario where the FlexVPN client acts as a relay for internal clients and sends all DNS requests for any domain to a server on the service provider network because the VPN tunnel to FlexVPN server is down.

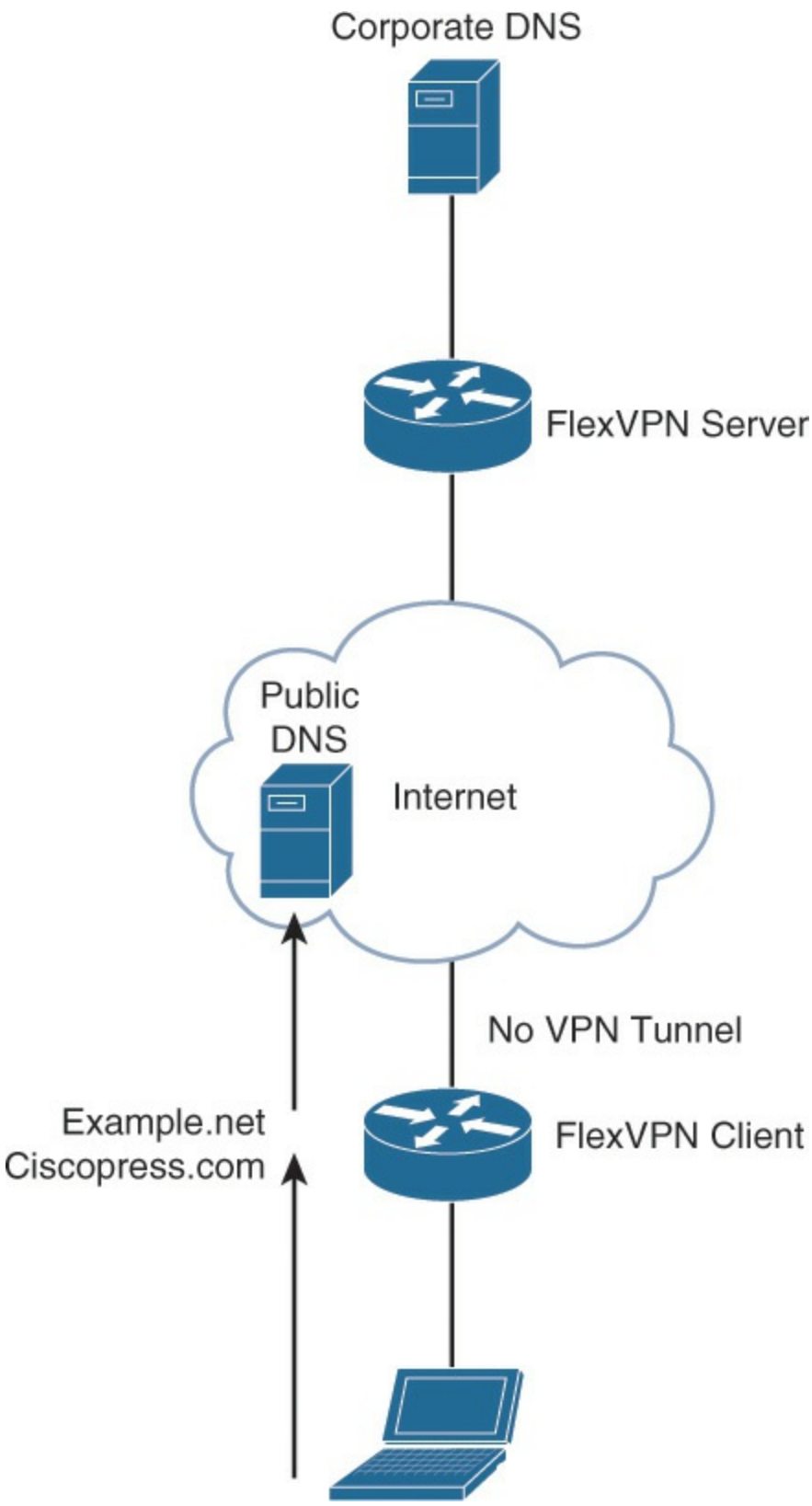


Figure 10-2 *FlexVPN Client Sending DNS Requests to Public DNS*

After the FlexVPN client establishes a VPN tunnel to the FlexVPN server, a DNS view is installed on the FlexVPN client, and requests for specific DNS domain queries (specified in the DNS view) are sent via the VPN.

Figure 10-3 illustrates split-DNS. The domain-name example.net is used for split-DNS and sent from the FlexVPN server and installed within a DNS view on the FlexVPN client to use the DNS server within the tunnel for domain. The FlexVPN client is acting as a DNS server for clients on the inside network. When these clients make requests that contain the example.net domain, these requests are sent to the DNS server through the tunnel. All other requests are sent to the DNS server on the service

provider network. This allows for the clients protected behind the FlexVPN client to use the Internet for all DNS requests apart from the domains local to the company. This saves Internet-bound traffic having to go from the FlexVPN client to the FlexVPN server and then on to the Internet.

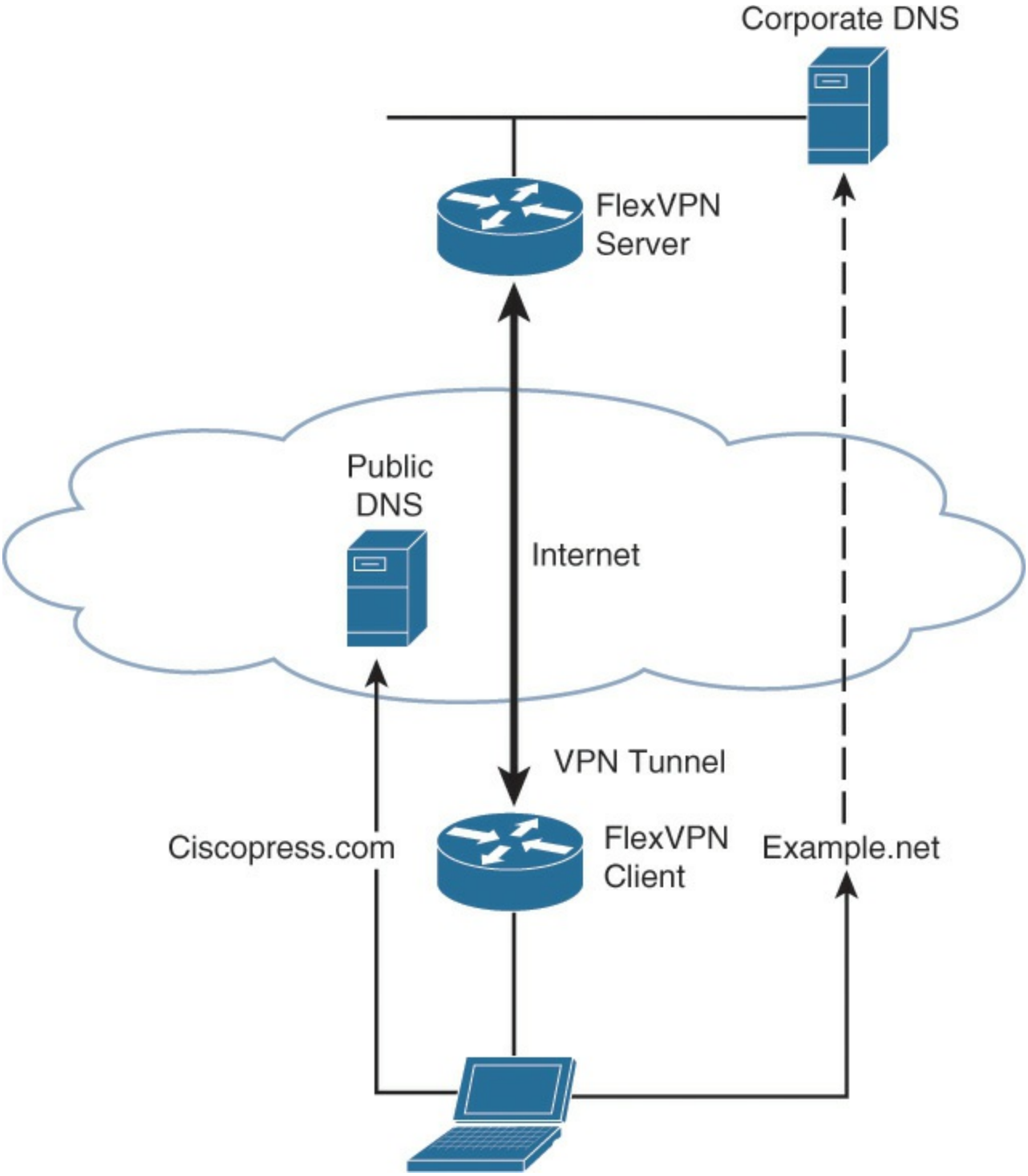


Figure 10-3 *FlexVPN Client Using Split-DNS*

Components of Split-DNS

Split-DNS requires the following components to be configured.

The FlexVPN server must have at least one DNS server configured as part of the AAA user or group authorization profile for the client either within the IKEv2 authorization policy for local AAA or on the RADIUS server. Up to two DNS servers can be configured for each IP address family (IPv4/IPv6) with the first acting as the primary and second as the secondary.

Table 10-1 illustrates the methods of configuring the IPv4 DNS server attributes.

IPv4 DNS Server Attribute	Configuration
RADIUS attribute type	Cisco AV pair
RADIUS configuration	**ipsec:dns-servers**=*ipv4-address1 [ipv4-address2]*
Configuration under IKEv2 authorization policy	**dns** *ipv4-address1 [ipv4-address2]*

Table 10-1 *Configuring IPv4 DNS Server Attributes*

Table 10-2 illustrates the methods of configuring the IPv6 DNS server attributes.

IPv6 DNS Server Attribute	Configuration
RADIUS attribute type	Cisco AV pair
RADIUS configuration	**ipsec:ipv6-dns-servers-addr**=*ipv6-address1 [ipv6-address2]*
Configuration under IKEv2 authorization policy	**ip6 dns** *ipv6-address1 [ipv6-address2]*

Table 10-2 *Configuring IPv6 DNS Server Attributes*

The FlexVPN server must be configured with the domains that will need to be proxied by the FlexVPN client. These domains must be configured as part of the AAA user or group authorization profile for the client, either within the IKEv2 authorization policy for local AAA or on the RADIUS server. Up to ten domains can be configured using a separate attribute or command for each.

Table 10-3 illustrates the methods of configuring the split-DNS attribute.

Split-DNS Attribute	Configuration
RADIUS attribute type	Cisco AV pair
RADIUS configuration	**ipsec:split-dns**=*domain-name*
Configuration under IKEv2 authorization policy	**split-dns** *domain-name*

Table 10-3 *Configuring Split-DNS Attributes*

The FlexVPN client must have a default DNS server configured along with a DNS view. Normally the DNS server will reside on the service provider network and the DNS Server pushed via the FlexVPN server will reside behind the FlexVPN tunnel.

The following example shows the relevant configuration on the FlexVPN client and server to enable split-DNS. After enabling DNS name-list debugging using the command **debug ip dns name-list** and DNS View debugging using the command **debug ip dns view**, the name list and DNS server can be seen being added to the local DNS view when the VPN become established. This can be verified by viewing the local DNS name list using the command **show ip dns name-list**, which will show domains received from the FlexVPN server. Before the VPN is established, this list is empty unless manually configured.

Configuration on FlexVPN server:

Click here to view code image

```
crypto ikev2 authorization policy default
 dns 192.168.10.5 192.168.10.6
 split-dns example.net
```

Configuration on FlexVPN client:

Click here to view code image

```
ip dns view default
 domain name-server 172.16.10.1
ip dns server
```

The following example illustrates the verification of a DNS view as it is installed on the FlexVPN client.

The selected **debug crypto ikev2** and **debug crypto ikev2 client flexvpn** output below capture the DNS server and split-DNS attributes received from the FlexVPN server.

Click here to view code image

```
Received Packet [From 10.10.10.2:500/To 10.10.10.1:500/VRF i0:f0]
Initiator SPI : F09A4EC10E7723C6 - Responder SPI : C980EE3D55880C04 Message id: 1
IKEv2 IKE_AUTH Exchange RESPONSE
Config-type: Config-reply
Attrib type: ipv4-dns, length: 4, data: 192.168.10.5
Attrib type: ipv4-dns, length: 4, data: 192.168.10.6
Attrib type: split-dns, length: 18, data: example.net

FlexVPN(flex1 : 80000009) DNS Primary: 192.168.10.5
FlexVPN(flex1 : 80000009) DNS Secondary: 192.168.10.6
FlexVPN(flex1 : 80000009) Split DNS Name: example.net
```

The **debug ip dns name** and **debug ip dns view** output below capture the creation of DNS view on FlexVPN client.

Click here to view code image

```
DNS_VIEW: creating view flexvpn-view-flex1
DNS_VIEW: Setting  dns forwarder 192.168.10.5 192.168.10.6 in view
flexvpn-view-flex1
DNS_VIEW: Setting  domain name-server 192.168.10.5 192.168.10.6 in view
flexvpn-view-flex1
DNS_NAMELIST: adding permit 'EXAMPLE.NET' to name-list 1
```

The **show crypto ikev2 sa detailed** command output displays the DNS server IP addresses pushed by the FlexVPN server.

Click here to view code image

```
FlexClient#show crypto ikev2 sa detailed
      DNS Primary: 192.168.10.5
      DNS Secondary: 192.168.10.6
```

The **show crypto ikev2 client flexvpn flex1 detail** command output displays the split DNS domains pushed by the FlexVPN server.

Click here to view code image

```
FlexClient#show crypto ikev2 client flexvpn flex1 detail
  Split DNS:
```

```
    example.net
```

The **show ip dns name-list** command output displays the split-DNS domains installed in the DNS view.

Click here to view code image

```
FlexClient#show ip dns name-list
ip dns name-list 1
    permit EXAMPLE.NET
```

Windows Internet Naming Service (WINS)

The FlexVPN server has the ability to push the Windows Internet naming service (WINS), also known as the NetBIOS name service (NBNS), using the IKEv2 configuration exchange, which is then integrated into the DHCP server running on the FlexVPN client. This allows for hosts that gain an IP address by the local DHCP server to have the configured WINS applied.

The FlexVPN server pushes the IPv4 WINS, using the standard IKEv2 configuration attribute INTERNAL_IP4_NBNS. At the time of writing, the FlexVPN server does not support pushing the IPv6 WINS, using INTERNAL_IP6_NBNS attribute.

Table 10-4 illustrates the methods of configuring the IPv4 WINS attribute.

IPv4 WINS Attribute	Configuration
RADIUS attribute type	Cisco AV pair
RADIUS configuration	**ipsec:wins-servers=***ipv4-address1 [ipv4-address2]*
Configuration under IKEv2 authorization policy	**wins** *ipv4-address1 [ipv4-address2]*

Table 10-4 *Configuring IPv4 WINS Attribute*

A DHCP server must be configured on the FlexVPN client that allows for DHCP options to be programmatically imported, this is achieved using the **import all** command within the DHCP server configuration. The WINS attributes can be verified that it has been passed in the IKEv2 exchange, using the command **show crypto ikev2 sa detailed**. To verify that the DHCP server has been updated can be achieved using the command **show ip dhcp import**.

The following example details the configuration on a FlexVPN server and FlexVPN client. The WINS server is configured as 10.10.210.10 on the FlexVPN server; this is passed to the FlexVPN client and dynamically updated on the local DHCP server.

Configuration on the FlexVPN server is as follows:

Click here to view code image

```
crypto ikev2 authorization policy default
 wins 10.10.210.10
```

Configuration on FlexVPN client:

Click here to view code image

```
ip dhcp pool pool1
 import all
 network 10.10.1.0 255.255.255.0
```

```
 default-router 10.10.1.1
 dns-server 10.10.1.1

crypto ikev2 client flexvpn flex1
  peer 1 172.16.10.1
  connect manual
  client inside Ethernet0/1
  client connect Tunnel1
```

The following example illustrates how the attributes can be verified that they have been updated to the local dynamic DHCP server. Devices that are allocated IP addresses from the DHCP server will have the dynamic attributes applied.

Click here to view code image

```
FlexVPNClient#show ip dhcp import
Address Pool Name: pool1
NetBIOS Name Server(s): 10.10.210.10

FlexClient#show crypto ikev2 sa detailed
      WINS Primary: 10.10.210.10
```

Domain Name

The FlexVPN server has the ability to push the default domain name, using the IKEv2 configuration exchange. This attribute is then integrated into the DHCP server running on the FlexVPN client and allows for the host client that gains an IP address by the local DHCP server to have the pushed domain-name applied.

The FlexVPN server pushes the default domain, using the Cisco proprietary IKEv2 configuration attribute MODECFG_DEFDOMAIN.

Table 10-5 illustrates the methods of configuring the default-domain attribute.

Default-Domain Attribute	Configuration
RADIUS attribute type	Cisco AV pair
RADIUS configuration	**ipsec:default-domain=***domain-name*
Configuration under IKEv2 authorization policy	**def-domain** *domain-name*

Table 10-5 *Configuring the Default-Domain Attribute*

A DHCP server must be configured on the FlexVPN client that allows for DHCP options to be programmatically imported. This is achieved using the command **import all** within the DHCP server configuration. The default domain name can be verified that it has been passed in the IKEv2 exchange, using the command **show crypto ikev2 sa detailed**. To verify that the DHCP server has been updated can be achieved using the command **show ip dhcp import**.

The following example details the configuration on a FlexVPN server and FlexVPN client. The default domain of **example.net** is configured on the FlexVPN server, passed to the FlexVPN client, and dynamically updated on the local DHCP server.

Configuration on the FlexVPN server:

Click here to view code image

```
crypto ikev2 authorization policy default
 wins 10.10.210.10
 def-domain example.net
```

Configuration on the FlexVPN client:

Click here to view code image

```
ip dhcp pool pool1
 import all
 network 10.10.1.0 255.255.255.0
 default-router 10.10.1
 dns-server 10.10.1.1

crypto ikev2 client flexvpn flex1
  peer 1 203.0.113.224
  connect manual
  client inside Ethernet0/1
  client connect Tunnel1

FlexVPNClient#show ip dhcp import
Address Pool Name: pool1
Domain Name Option: example.net

FlexClient#show crypto ikev2 sa detailed
      Default Domain: example.net
```

FlexVPN Client Profile

The FlexVPN client consists of a number of configuration building blocks that are required to connect to the FlexVPN server. The FlexVPN client profile is the inner workings of the FlexVPN client and defines the following

- Which FlexVPN server(s) the client will connect to
- The tunnel interface used to connect
- The method that the client will use to connect (auto/manual/track)
- The source interface used to initiate the tunnel
- Backup groups

These attributes are configured within the client profile, which is configured using the command **crypto ikev2 client flexvpn** *profile-name***.**

The client profile will integrate with the other FlexVPN components to dynamically build a tunnel used to connect to the FlexVPN server, rather than having a statically configured tunnel with the IP address of the FlexVPN server. The FlexVPN client must have the matching cryptographic attributes as the FlexVPN server configured in the IKEv2 proposal. Like any other implementation of FlexVPN, an IKEv2 profile will be used to define the IKEv2 identity, VRF, match peer identity, and so on.

Backup Gateways

Multiple peers can be configured on the FlexVPN client for each FlexVPN server that the client will attempt to connect to, although only one peer will be active at any given time. Peers are ordered in a sequential manner with the lower-sequenced peer having the higher preference. Connections will be attempted to peers in order; if the connection to one peer fails, the next peer will be attempted. Four IKEv2 connections will be attempted, with an incrementing back-off timer between connection attempts. The back-off timer will start at 2 seconds, then double to 4, 8, and then 16 seconds.

The peers are configured in the FlexVPN client profile using the following command.

Click here to view code image

```
peer sequence {ipv4-address | ipv6-address | fqdn fqdn-name [dynamic | ipv6]}
[track track-number [up | down]]
```

Resolution of Fully Qualified Domain Names

If the peer is defined by FQDN, by default this will be resolved to an IP address by DNS when the configuration is applied unless the **dynamic** keyword is included. If the **dynamic** keyword is used, the FQDN will be resolved to an IP address using DNS when the tunnel is initiated. This name resolution will be performed only once; as soon as the FQDN is resolved, the corresponding IP address will be added to the configuration. If the configuration is then saved (using the **copy run start** command), the saved configuration will contain the destination of an IP address and not FQDN.

Reactivating Peers

Should a connection successfully establish to a peer that doesn't have the highest preference, by default if a more preferential peer should then become available, the connection to the most preferential peer will not be established. This behavior can be changed using the **peer reactivate** command, which will enable the current connection to terminate and a new connection to be attempted with the more preferred peer.

Backup Gateway List

In addition to the manually configured peers on a FlexVPN client, a FlexVPN server can push a backup gateway list during the session bring up, using IKEv2 configuration exchange, which is a list of additional peers that the client can connect to, should the connection to the server fail. The downloaded list is inserted right after the peer from which the list was downloaded.

The FlexVPN server can push up to 10 backup gateways using the Cisco proprietary IKEv2 configuration attribute MODECFG_BACKUPSERVERS.

Table 10-6 illustrates the methods of configuring the backup-gateway attribute.

Backup-Gateway Attribute	Configuration
RADIUS attribute type	Cisco AV pair
RADIUS configuration	`ipsec:ipsec-backup-gateway=`*`ip-address	hostname*
Configuration under IKEv2 authorization policy	**`backup-gateway`** {*`ip-address	hostname`*}

Table 10-6 *Configuring Backup-Gateway Attribute*

The following example illustrates the backup gateway list functionality.

A client is configured with three static peers (A, B, C) in the listed order of priority. For any reason, when the client connects to peer B and a list of three other peers (1, 2, and 3) is downloaded from that peer, the complete ordered gateway list effectively becomes

- Peer A (priority 1)
- Peer B (priority 2)
  - Peer B.1 (priority 2.1)
  - Peer B.2 (priority 2.2)
  - Peer B.3 (priority 2.3)
- Peer C (priority 3)

In this case, the static peer B is defined as the parent peer for the downloaded list. If peer B.1, B.2, or B.3 pushed a new list at connection time, the old downloaded list would be deleted and replaced with the new list, keeping B as a parent peer for the downloaded list.

Tunnel Interface

The FlexVPN client uses a static point-to-point tunnel interface with either GRE or native IPsec encapsulation mode to connect to the FlexVPN server. The tunnel interface used by FlexVPN client is unique in the following ways.

The IP address of the tunnel interface can either be statically configured or be configured to request an IP address from the FlexVPN server using the **ip address negotiated** command.

The tunnel source and tunnel destination can be dynamically programmed from the FlexVPN client profile, based on the availability of the peer and the source interface which is enabled using **tunnel source dynamic** and **tunnel destination dynamic** commands. Note that when using dynamic tunnel source and destination, the currently active tunnel source and destination can be seen in the **show crypto ikev2 client flexvpn** command output and not in the running configuration.

The tunnel interface is referenced from the client profile, using the command **client connect tunnel interface-number**. The tunnel destination address is dynamically obtained from the client profile; this IP address is generated from the selected peer statement in the client profile. The tunnel interface must be protected with an IPsec profile referencing the IKEv2 profile, using the command **crypto IPsec profile** *name*.

Tunnel Source

The source interface of the tunnel can be configured to be static or dynamic. If the tunnel source will always use the same interface, then the tunnel source can be statically set using the command **tunnel source** *interface|ip-address*.

If multiple source interfaces are needed for the client the tunnel interface must be set to dynamic using the command **tunnel source dynamic.** If the tunnel source is set to dynamic, the tunnel source will be obtained from the client profile. The client profile can be configured with multiple interfaces that can be potentially used as tunnel source, with each interface assigned a unique sequence number and a track object. An interface is considered available only when the track object is in the "up" state. The available interface with the lowest sequence number and with the VRF matching the "tunnel fvrf" is used as tunnel source. If a session is "up" with a given source then, such a source is said to be a "current active source." Any change in current active source will terminate the active session.

Using a dynamic tunnel source allows for multiple bearers to be used as the transport network. Many customer will use a single device for secure connectivity. This device will have a number of connections to the VPN headend, each using different transport networks of varying quality and cost. Having a dynamic tunnel source allows for the choice of the higher quality or less costly bearer to be used when the VPN headend is reachable over this link. If connectivity to the VPN headend fails, the backup link sources the traffic from a different interface. This allows for costly connections to only be used when required, resulting in savings when backup links are charged, based on the amount of traffic sent.

The following example illustrates the dynamic tunnel source configuration. The current value of the tunnel source can be seen in the **show crypto ikev2 client flexvpn** command output's **source** field as highlighted below. Note that the running configuration does not display the current tunnel source interface.

Click here to view code image

```
interface Tunnel10
 ip address negotiated
 tunnel source dynamic

crypto ikev2 client flexvpn flex2
  source 1 Ethernet0/0 track 1
  source 2 Ethernet0/1 track 2
  client connect Tunnel10

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex2
  Current state:ACTIVE
  Peer : 10.0.0.2
  Source : Ethernet0/0
```

Tunnel Destination

The tunnel destination can be configured to be static or dynamic. If there is a single peer then the tunnel destination can be statically set using the command **tunnel destination** *ip-address*.

If multiple peers are available, the tunnel destination can be configured to be dynamically set using the **tunnel destination dynamic** command. The FlexVPN client profile can be configured with multiple peers, each with a unique sequence number and optionally a track object. The peers with a

track object are considered available when the associated track object is in the desired state, and the peers without a track object are considered always available. The client profile dynamically programs the tunnel destination to the peer listed in the client profile that has the least sequence number and is available. The current value of the tunnel destination can be seen in the **show crypto ikev2 client flexvpn** command output's **peer** field as highlighted below. Note that the running configuration does not display the current tunnel destination value. The following example illustrates the dynamic tunnel destination configuration.

Click here to view code image

```
crypto ikev2 client flexvpn flex2
  peer 1 10.0.0.2 track 1
  peer 2 10.0.0.3 track 2 down
  client connect Tunnel10

interface Tunnel10
 ip address negotiated
 tunnel destination dynamic

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex2
  Current state:ACTIVE
  Peer : 10.0.0.2
```

Tunnel Initiation

The FlexVPN client can initiate the VPN using three modes, this is configured within the client profile using the command **connect** {**auto** | **manual** | **track** *number*}.

Automatic Mode

Automatic mode is used to automatically start the FlexVPN tunnel activation. If the FlexVPN client is correctly configured, the IKEv2 exchange will begin. User intervention is not required once this mode is enabled. This is the default mode.

Manual Mode

Manual mode requires a user (or user-configured script) to initiate the FlexVPN tunnel using the **crypto ikev2 client flexvpn connect *name*** command in exec mode. Once established, the tunnel can be removed using the clear **crypto ikev2 client flexvpn *name*** command in exec mode. This command will only initiate a single attempt to a peer; if the peer in unavailable, then the IKEv2 negotiation will fail, and the next connection attempt will proceed to the next configured peer. However, this connection will not be attempted until the user initiates it.

Track Mode

The FlexVPN client can initiate a connection, depending on the state of a tracked object. The client process integrates to the enhanced object tracking feature and receives notifications when the state of tracked object changes.

The enhanced object tracking feature allows for a tracking of objects, such as:

- The line protocol state on an interface

- The presence of IP routing that is enabled on an interface
- The state on an SLA operation
- The presence of a prefix in the routing table
- The reachability of an SLA host

Tracking a List of Objects, Using a Boolean Expression

The tracked object can either be in the "up" or "down" state. If the track keyword is set to activate the tunnel when the object is in the "up" state, the client triggers the connection upon receiving a notification that the object is in the "up" state. If the track keyword is set to activate the tunnel when the object is in the "down" state, the client triggers the connection upon receiving the notification that the object is in the "down" state.

The following example shows the connect track command being used to only activate the tunnel should the line protocol on interface Ethernet 0/2 be "up"; if the line protocol is "down," the tunnel will not activate. Should a FlexVPN client establish a connection to the FlexVPN server and the status of the track event change from "up" to "down," the tunnel to the FlexVPN server will be brought down.

Define the track object.

Click here to view code image

```
track 10 interface Ethernet0/2 line-protocol
```

Reference the track object from the client profile.

Click here to view code image

```
crypto ikev2 client flexvpn flex1
  peer 1 203.0.113.225
  connect track 10
  client inside Ethernet0/1
  client connect Tunnel1
```

The following example shows the track state changing status; in turn, this brings the FlexVPN client up or down, depending on the tracked event.

Click here to view code image

```
%TRACKING-5-STATE: 10 interface Et0/2 line-protocol Down->Up
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel1, changed state to up

%TRACKING-5-STATE: 10 interface Et0/2 line-protocol Up->Down
%FLEXVPN-6-FLEXVPN_CONNECTION_DOWN: FlexVPN(flex1) Client_public_addr =
209.165.201.1 Server_public_addr = 203.0.113.225
```

Using tracked events can allow for scenarios with very many options. This is due to the fact that a single track event can track multiple other tracked events using a boolean or threshold outcome for events. Examples include a series of multiple events occurring using the boolean AND operation, along with the occurrence of only one of a number of other events, using the boolean OR operation.

In the following example, a number of track commands are used. First, an IP SLA is created to check whether a device is reachable using the ICMP echo function. Then a track will monitor the status of this IP SLA using track 1. The presence of a prefix (209.165.201.8) in the routing table is tracked

using track 2. Track 3 monitors the status of the line protocol of a local interface. Then another tracked object (track 4), is used as a boolean expression to bind all three elements. We actually want to only allow this track to be "up" should both track 1 and 2 be "up" and track 3 be "down." We achieve this using the boolean NOT command when adding track object 3.

Click here to view code image

```
ip sla 1
 icmp-echo 203.0.113.225
ip sla schedule 1 life forever start-time now

track 1 ip sla 1

track 2 ip route 209.165.201.8 255.255.255.255 reachability

track 3 interface Ethernet0/2 line-protocol

track 4 list boolean and
 object 1
 object 2
 object 3 not

FlexClient#show track
Track 1
  IP SLA 1 state
  State is Up
    1 change, last change 01:30:49
  Latest operation return code: OK
  Latest RTT (millisecs) 1
  Tracked by:
    Track-list 4
Track 2
  IP route 1 209.165.201.8 255.255.255.255 reachability
  Reachability is Up (static)
    1 change, last change 00:27:46
  First-hop interface is Ethernet0/0
  Tracked by:
    Track-list 4
Track 3
  Interface Ethernet0/2 line-protocol
  Line protocol is Down (hw admin-down)
    2 changes, last change 00:00:08
  Tracked by:
    Track-list 4
     0
Track 4
  List boolean and
  Boolean AND is Up
    4 changes, last change 00:00:07
    object 1 Up
    object 2 Up
    object 3 not Down
```

Dial Backup

In scenarios where the FlexVPN client router has multiple WAN connections, the track-based tunnel activation feature can be used to implement the dial backup feature (aka primary and backup tunnels) in the following ways.

- A FlexVPN tunnel can be created over each of the WAN interfaces with one tunnel acting as primary and the other as backup. This can be accomplished by having both the tunnels track the same object for tunnel activation with one tunnel tracking the "up" state and the other tracking the "down" state of the common track object.
- A FlexVPN tunnel can be created over the one of the WAN interfaces (e.g., Internet) that acts as a backup to another WAN connection (e.g., dial-up connection). This can be accomplished by using track-based tunnel activation on the FlexVPN tunnel tracking the primary connection going down.

Backup Group

Each FlexVPN client profile can be manually assigned to a backup group; if a backup group is not configured, the default backup group of "0" or default is used. A configuration profile can be configured to participate in a backup group with the **backup group** *1-255* command.

When a FlexVPN client is required to initiate a connection to a peer, it will validate if there are any other FlexVPN client profiles in the same backup group which have established sessions to the same peer. If a connection is established and the client profile is a member of the same backup group, then the connection to the peer will be bypassed and the next peer attempted.

If the same peer is configured in separate profiles with different backup groups configured, there is a possibility that a simultaneous connection will be made to both peers.

In the following example, two FlexVPN client profiles are configured with the same backup group (10). The FlexVPN client profile named "flex1" establishes a connection to peer 172.16.1.1. The FlexVPN client profile named "flex2" fails to establish a connection to the first configured peer (172.16.2.1). After the IKEv2 timeout, the second peer is tried. Because there is already an active connection to this peer using the FlexVPN client profile "flex1" and these are in the same backup group, this peer is skipped, and a connection to the next peer is attempted. The syslog messages were generated after enabling FlexVPN client debugging, using the command **debug crypto ikev2 client flexvpn**.

Click here to view code image

```
crypto ikev2 client flexvpn flex1
  peer 1 172.16.1.1
  backup group 10
  client connect Tunnel1

crypto ikev2 client flexvpn flex2
  peer 1 172.16.2.1
  peer 2 172.16.1.1
  peer 3 172.16.3.1
  backup group 10
  client connect Tunnel2

FlexVPN(flex2 : 8000000B) Current_state: NEGOTIATING
FlexVPN(flex2 : 8000000B) Current_event: EV_TP_ERROR
FlexVPN(flex2 : 8000000B) Error during negotiation
initiating auto reconnect timer
FlexVPN(flex2 : 8000000B) Current_state: NEGOTIATING
FlexVPN(flex2 : 8000000B) Current_event: EV_DISCONNECT
%FLEXVPN-6-FLEXVPN_CONNECTION_DOWN: FlexVPN(flex2) Client_public_addr = 172.16.4.1
Server_public_addr = 172.16.2.1
```

```
FlexVPN(flex2 : 0) Connection being terminated with peer 198.51.100.133
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel2, changed state to down
FlexVPN(flex2 : 0) advanced to next peer 172.16.1.1
FlexVPN(flex2 : 0) Current_state: CONNECT_REQUIRED
FlexVPN(flex2 : 0) Current_event: EV_CONNECT
FlexVPN(flex2 : 0) Current_state: CONNECT_REQUIRED
FlexVPN(flex2 : 0) Current_event: EV_SET_PEER
FlexVPN(flex2 : 0) Validating peer 172.16.1.1
FlexVPN(flex2 : 0) Cannot connect to peer 172.16.1.1 since FlexVPN (flex1) is
already connected
FlexVPN(flex2 : 0) Validating peer 172.16.3.1
FlexVPN(flex2 : 0) Ready to connect to peer 172.16.3.1
FlexVPN(flex2 : 0) Current peer set to 172.16.3.1
```

Network Address Translation

Although not specifically a FlexVPN client feature, Network Address Translation (NAT) requires mentioning because it goes hand-in-hand with the FlexVPN client feature. The FlexVPN client supports two scenarios where NAT can be used.

If the FlexVPN client is to be used to protect devices on an "inside" network, which uses non-routable (RFC1918) IP addresses, and the FlexVPN client is to provide Internet services to these devices, then NAT can be used to allow the devices to be translated to an Internet-routable IP address when passing through the router.

If the FlexVPN client is to be used to protect devices on an "inside" network that uses IP addresses that are not in the routing table of the FlexVPN server, then NAT can be used to translate IP addresses to the IP address of the tunnel interface when traffic is sent via the point-to-point tunnel between the client and server.

The following example illustrates the configuration on a FlexVPN client that allows clients on the inside network that gain their address via DHCP to be port address translated (PAT'd) to the IP address that is assigned to the tunnel interface.

Click here to view code image

```
ip dhcp pool pool1
 network 172.16.1.0 255.255.255.0

interface Tunnel1
 ip address negotiated
 ip nat outside
 ip virtual-reassembly in
 tunnel source Ethernet0/0
 tunnel destination dynamic
 tunnel protection ipsec profile default

interface Ethernet0/1
 ip address 172.16.1.1 255.255.255.0
 ip nat inside
 ip virtual-reassembly in

access-list 1 permit 172.16.1.0 0.0.0.255

ip nat inside source list 1 interface Tunnel1 overload

FlexClient#show ip nat translations
Pro Inside global      Inside local       Outside local      Outside global
```

```
icmp 192.168.1.1:6     172.16.1.2:6       192.168.2.1:6      192.168.2.1:6
tcp 192.168.1.1:22079  172.16.1.2:22079   192.168.2.1:23     192.168.2.1:23
```

The following example illustrates NAT configuration required to translate the enterprise traffic to the server-assigned IP address and split tunnel Internet-bound traffic to a globally routable IP address. The **ip nat outside** command is configured on tunnel and outbound interfaces and route maps are configured to allow for NAT rules based on the NAT outside interface.

Click here to view code image

```
interface Tunnel 0
 description FlexVPN client tunnel interface
 ip address negotiated
 ip nat outside

interface Ethernet 0/0
 description WAN-interface
 ip nat outside

interface Ethernet 1/0
 description LAN-interface
 ip nat inside

route-map internet-route permit 1
 match interface Ethernet 0/0

route-map flexvpn-route permit 1
 match interface Tunnel 0

ip nat inside source route-map crypto-route interface Tunnel0 overload
ip nat inside source route-map internet-route interface Ethernet0/0 overload
```

Design Considerations

Use of Public Key Infrastructure and Pre-Shared Keys

Although implementing a PKI is preferred over using pre-shared-keys, the use of implementing both pre-shared key and certificate authentication can work well when deploying a FlexVPN client solution. The FlexVPN server(s) can be enrolled into a PKI, while all clients can be configured with a trustpoint containing the PKI used on the FlexVPN server. The client will authenticate the server using the certificate of the server to ensure authentication, and the server will authenticate the client using a pre-shared key. By implementing a RADIUS server containing the pre-shared keys for all clients and allowing the server to use RADIUS to obtain the pre-shared keys for clients, authentication using pre-shared keys for clients can be easily managed from a central location.

The Power of Tracking

The following scenario is where track mode was a perfect fit. A solution was required for a team of mobile engineers that required VPN access over a 3G network. The engineers only wanted to use the VPN when they had their diagnostic IP device connected to the portable router because there was a cost involved when using the 3G network. Because the inside interface would be in the “up” state when the diagnostic device was powered on, a track was used to monitor when the inside interface was in the “up” state and then linked to the FlexVPN client. The VPN would then be initiated only when the diagnostic IP device was connected.

Tracked Object Based on Embedded Event Manager

Embedded event manager (EEM) is a distributed and customized approach to event detection and recovery offered directly in Cisco IOS and IOS-XE devices. EEM offers the ability to monitor events and take informational, corrective, or any desired EEM action when the monitored events occur or when a threshold is reached. An EEM policy is an entity that defines an event and the actions to be taken when that event occurs.

Although EEM does not directly influence the creation of FlexVPN client tunnels, EEM events can be used to influence the status of tracked objects allowing for FlexVPN client tunnels to be initiated, depending on the status of a number of complex events.

A customer approached Cisco Advanced Services to assist in the migration of a VPN solution where routers with multiple WAN interfaces (one having a private, RFC1918 IP address, and the other having an Internet routable IP address) were to be replaced by a router model with a single WAN interface.

When using the router with two WAN interfaces, the customer had multiple tunnel interfaces configured, with each tunnel having the source of each WAN interface, but the new router model only allowed them to connect to one service provider network at a time. This configuration led to a tunnel interface that was connected when the physical WAN interface was active. The customer needed a solution where, if the WAN interface was connected to a private network (and gained a RFC1918 IP address), a tunnel would be initiated to a VPN headend on the private network. If the IP address on the WAN interface was not a RFC1918 address, then the tunnel would initiate to a headend on the Internet. To achieve this, an EEM script was developed to detect the presence of a RFC1918 IP address on the WAN interface; if this occurred, then a tracked object (track 1) would be set to "up." Another tracked object (track 2) would be set as the inverse for track 1. A second EEM script was written to detect if the WAN interface went down, this would set track 1 to "down." The FlexVPN client profile would use the peer on the private network which would use track 1, while the peer on the Internet would use track 2. The following example illustrates the configuration and EEM script used to accomplish this.

Click here to view code image

```
track 1 stub-object

track 2 list boolean and
 object 1 not

event manager applet intDown
 event syslog pattern "Interface Ethernet0/0, changed state to down"
 action 1 track set 1 state down

event manager applet getIP
 event syslog pattern "Interface Ethernet0/0, changed state to up"
 action 1.0 cli command "enable"
 action 1.1 cli command "show int e0/0 | inc address is [0-9\.]+"
 action 2.0 regexp "address is ([0-9\.]+)" "$_cli_result" match ip
 action 2.1 regexp "(^[0-9]+)\.([0-9]+)" "$ip" match first second
 action 3.0 if $first eq "10"
 action 3.1  syslog msg "This is a RFC 1918 address $ip"
 action 3.2  track set 1 state up
 action 3.3 end
 action 4.0 if $first eq "192"
```

```
 action 4.1  if $second eq "168"
 action 4.2   syslog msg "This is a RFC 1918 address $ip"
 action 4.3   track set 1 state up
 action 4.5  end
 action 4.6 end
 action 5.0 if $first eq "172"
 action 5.1  if $second ge "16"
 action 5.2   if $second le "31"
 action 5.3    syslog msg "This is a RFC 1918 address $ip"
 action 5.4    track set 1 state up
 action 5.5   end
 action 5.6  end
 action 5.7 end

crypto ikev2 client flexvpn flex1
  peer 1 172.16.1.3 track 1
  peer 2 192.168.1.2 track 2
  client connect Tunnel1
```

Troubleshooting FlexVPN Client

When deploying the FlexVPN client, there are a number of tools that can be used, which are specific to FlexVPN client.

Useful Show Commands

The command **show crypto ikev2 client flexvpn** displays information about the configured client profiles, the status of each profile, and if connected, downloaded attributes. A specific client profile can be shown by including the name of the profile, using the command **show crypto ikev2 client flexvpn name**.

The following example displays information about the configured client profiles, the first profile (flex_client_profile_1) is active and has an allocated IP address. The second profile (flex_client_profile_2) is configured to be connected manually; this connection has not been initiated, which can be seen from the current state being CONNECT_REQUIRED.

Click here to view code image

```
FlexClient#show crypto ikev2 client flexvpn
  Profile : flex_client_profile_1
  Current state:ACTIVE
  Peer : 203.0.113.225
  Source : Ethernet0/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel1
  Assigned IP address: 10.0.0.2

  Profile : flex_client_profile_2
  Current state:CONNECT_REQUIRED
  Backup group: Default
  Tunnel interface : Tunnel2
```

The following lists the attributes displayed by the command **show crypto ikev2 client flexvpn**.

- FlexVPN profile name

- Current active peer
- Current active source
- IVRF and FVRF
- Backup group
- Tunnel interface
- Peers
- Assigned IP address
- Source interface
- Current state

The following example illustrates the **show crypto ikev2 sa detailed** command. This command will display the attributes pushed by FlexVPN server to the client.

Click here to view code image

```
FlexClient#show crypto ikev2 sa detailed
      DNS Primary: 192.168.10.5
      DNS Secondary: 192.168.10.6
      WINS Primary: 10.10.210.10
      Default Domain: example.net
```

The **show track** command can be used to verify if a tracked event is in the "up" or "down" state; this can be useful should a FlexVPN client session be in tracked mode.

The following example shows two configured track commands and the verification that these are in the desired state. The presence of FlexVPN within the "Tracked by" indicates that this is used by a FlexVPN client profile.

Click here to view code image

```
FlexClient#show track
Track 1
  Stub-object
  State is Up
    10 changes, last change 02:23:21, by EEM
  Tracked by:
    Track List 2
    FlexVPN 0
Track 2
  List boolean and
  Boolean AND is Down
    11 changes, last change 02:23:20
    object 1 not Up
  Tracked by:
    FlexVPN 0
```

Debugging FlexVPN Client

The command **debug crypto ikev2 client flexvpn** will enable debugging messages that relate to the operation of the FlexVPN client and how it interacts with other processes. The logging messages will display events and give a view of how the client profile calls other processes, such as the tunnel interface, and how the peer is selected. There are no specific details displayed in the logging messages for attributes passed using mode-config from the FlexVPN server. The output from the logging messages generated is sparse compared to the output generated by the **debug crypto ikev2** command, and for this reason, this command can be handy in diagnosing client connections from a high level.

To enable viewing of the attributes passed in the IKEv2 exchange, the command **debug crypto ikev2** can be used to enable debugging. Additionally the command **debug crypto ikev2 packet** or **debug crypto ikev2 packet hexdump** can be used for more detailed output.

The following example details the partial output from a FlexVPN client having **debug crypto ikev2 packet hexdump** enabled

Click here to view code image

```
VID  Next payload: VID, reserved: 0x0, length: 23
     43 49 53 43 4F 2D 44 45 4C 45 54 45 2D 52 45 41
     53 4F 4E

 VID  Next payload: NOTIFY, reserved: 0x0, length: 21
     46 4C 45 58 56 50 4E 2D 53 55 50 50 4F 52 54 45
     44
```

The above output shows the Vendor ID payload containing the hexadecimal representation of the strings CISCO-DELETE-REASON and FLEXVPN-SUPPORTED, indicating that FlexVPN client is in use.

Clearing IKEv2 FlexVPN Client Sessions

The command **clear crypto ikev2 client flexvpn** will clear all the current FlexVPN client connections. Specific profiles can be cleared by using the **clear crypto ikev2 client flexvpn *name*** command, including the name of the profile. When clearing the client connection for a profile that uses a manual connection **(connect manual)**, the IKEv2 SA will be cleared, bringing down the IPsec Security Association. When clearing the client connection for a profile that uses an auto connection **(connect auto)** or tracked connection **(connect track)**, the IPsec Security Association will be recreated, that is a CREATE_CHILD_SA exchange will occur, keeping the current IKEv2 SA.

To clear the IKEv2 SA for the peer, use the command **clear crypto ikev2 sa remote** [*ipv4 address* | *ipv6 address*].

Summary

FlexVPN client allows for remote working for small-office, home-office, and mobile workers, such as salespeople. Using the power of mode-config, attributes can be passed in the IKEv2 exchange that allows configuration on the FlexVPN client that integrates with local features. The FlexVPN client is designed to provide secure communications from the remote location to the FlexVPN server; the client will provide services such as DHCP and DNS to other devices connected to the router. The attributes passed from the FlexVPN server will integrate with these local services dynamically, allowing local devices to receive configuration from the FlexVPN client via DHCP and DNS that was passed from the FlexVPN server.

The benefits are that FlexVPN client can be used in virtually any scenario. The features implemented using FlexVPN client are not just for mobile workers, they can be implement whenever FlexVPN is used.

Chapter 11. FlexVPN Load Balancer

Introduction

The FlexVPN Load Balancer feature provides the ability to cluster a number of FlexVPN gateways and distributes incoming IKEv2 connection requests evenly among each of these gateways. Using IKEv2 redirects the incoming client requests are redirected to the least loaded FlexVPN gateway based on the system and cryptographic load. This allows for a scalable pay-as-you-grow architecture where VPN gateways can be added or removed from the cluster, depending on the number of sessions required.

Components of the FlexVPN Load Balancer

The FlexVPN load balancer is constructed of a number of components. By understanding those components, you will have a great understanding of the overarching feature.

IKEv2 Redirect

RFC5685 “Redirect Mechanism for the Internet Key Exchange Protocol Version 2 (IKEv2)” describes a method to redirect an IKEv2 client from one IKEv2 gateway to another IKEv2 gateway.

This behavior allows IKEv2 clients to connect to a single IP address or FQDN as the VPN headend and then allows the VPN gateway owning this IP address to redirect the client to another gateway to provide load balancing functionality.

An IKEv2 client will indicate the ability to perform the IKEv2 redirect mechanism and its willingness to be redirected by including a REDIRECT_SUPPORTED notification message within the IKE_SA_INIT exchange. On receipt of this indication, the IKEv2 gateway can then redirect the IKEv2 client to another IKEv2 gateway of its choosing.

The redirection is performed, using the REDIRECT notification message. This notification includes information about the IKEv2 gateway used to redirect the client to and the nonce data received from the IKEv2 client in the IKE_SA_INIT message. The redirection can be performed by the IKEv2 gateway in the IKE_SA_INIT reply message or in the IKE_AUTH message.

If the redirect is performed within the IKE_SA_INIT exchange, then the computationally expensive Diffie-Hellman key generation is not performed, however no authentication is performed by the IKEv2 gateway with regard to the client. If the redirect is performed in the IKE_AUTH exchange, then both peers authenticate each other prior to performing the redirect. This allows the IKEv2 gateway to ensure that only legitimate clients are redirected and that the client can validate it has been redirected from the legitimate gateway. However, this comes with the cryptographically expensive operations involved with authentication.

Figure 11-1 illustrates the IKE exchange when a redirect occurs within the IKE_SA_INIT exchange.

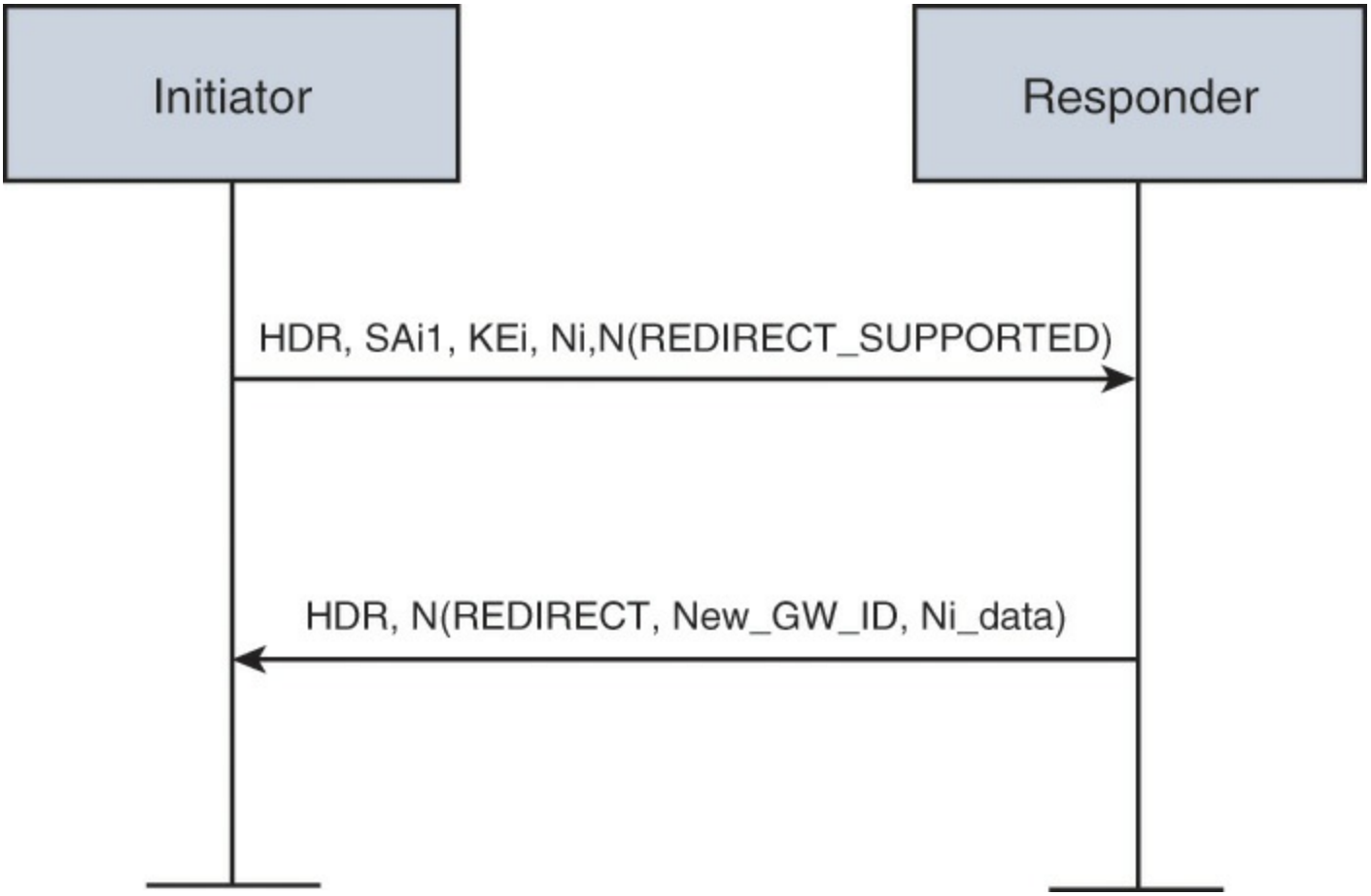


Figure 11-1 *IKE_SA_INIT with Redirect*

Figure 11-2 illustrates the IKE exchange when a redirect occurs within the IKE_AUTH exchange.

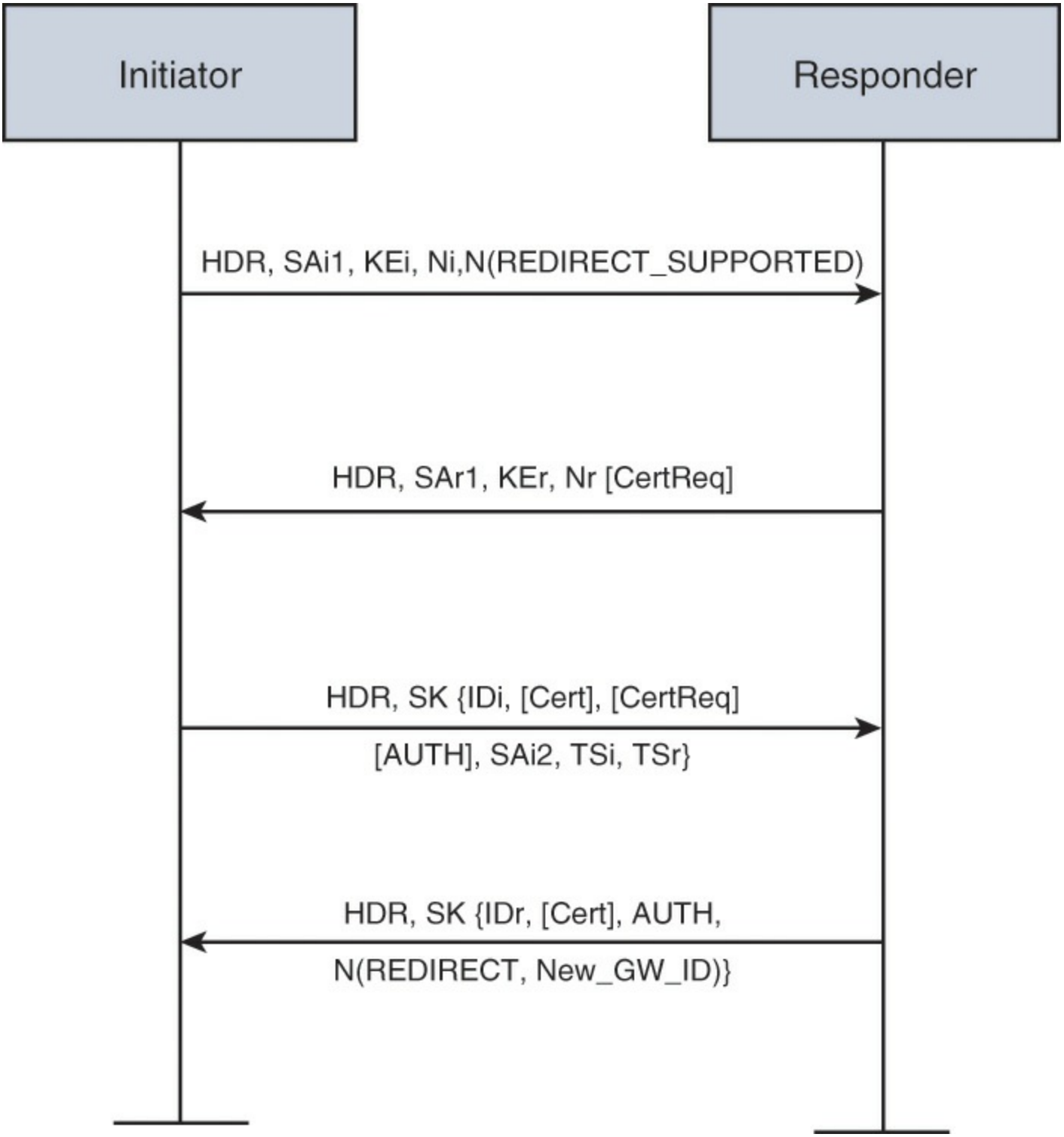


Figure 11-2 *IKE_AUTH with Redirect*

The inclusion of the nonce data within the REDIRECT notification ensures that an unsolicited forged reply cannot be used by an attacker to redirect clients to IKEv2 gateways of their choosing. because the nonce data is chosen randomly, an out-of-band attacker would be required to guess this value making this attack computationally infeasible.

When a client receives the REDIRECT notification, assuming that the nonce value matched the one that was sent originally the client will have all the information required to connect to the new IKEv2 gateway. This exchange will then occur; however, a REDIRECTED_FROM notify payload will be included which contains the IP address of the gateway that redirected the client.

Figure 11-3 illustrates the IKE_SA_INIT exchange once a client has been redirected from a gateway.

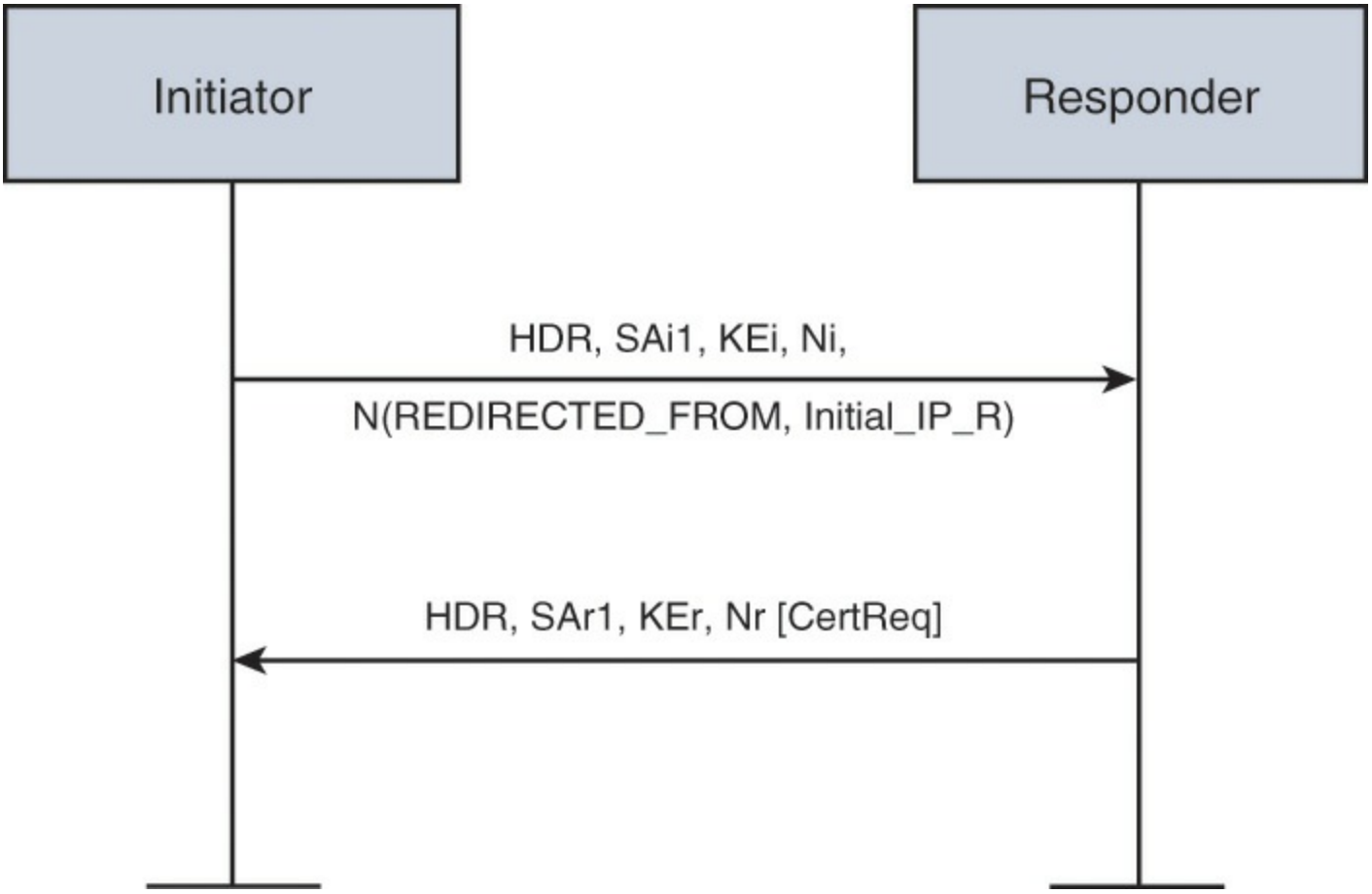


Figure 11-3 *IKE_SA_INIT Post Redirection*

Hot Standby Routing Protocol

The role of the hot standby routing protocol (HSRP) is to make a group of layer 2 adjacent routers appear as a single virtual router. This is achieved by using a virtual IP address (VIP), which is an IP address that belongs to the HSRP group and is owned by a single physical router at any given time. This physical router, known as the active router, will receive all traffic sent to the VIP and send traffic with the VIP as the source IP address. The VIP has a related virtual MAC address. The other physical routers, known as standby routers, are not actively passing traffic but maintain HSRP state with the active router. When the active router fails, a standby router automatically takes over the active role and begins sending and receiving traffic, using the VIP. All routers in the HSRP group must have an IP address on the local layer 2 network that is separate from the VIP.

> **Note**
> HSRP is documented in RFC2281 "Cisco Hot Standby Router Protocol (HSRP)."

The mechanism used to select the master is achieved using a priority value, which is user configurable and allows for the value between 1 and 255, with the higher value being preferred. In the case of two routers having the same priority, the numerical IP address would be the differentiator.

The following example illustrates HSRP configured on two routers using the VIP of 10.10.10.100 with a priority of 255 for router1 and 200 for router2 (the default being 100). The standby group name is defined as "group1". With this configuration, router1 will be the HSRP master, and router2 will be the HSRP standby.

Click here to view code image

```
Router1#show run interface Ethernet0/0
interface Ethernet0/0
 ip address 10.10.10.1 255.255.255.0
 standby 1 ip 10.10.10.100
 standby 1 priority 255
 standby 1 name group1

Router2#show run interface Ethernet0/0
interface Ethernet0/0
 ip address 10.10.10.2 255.255.255.0
 standby 1 ip 10.10.10.100
 standby 1 priority 200
 standby 1 name group1
```

FlexVPN IKEv2 Load Balancer

FlexVPN IKEv2 Load Balancer introduces the ability to cluster a number of FlexVPN servers together to terminate IKEv2 SAs from FlexVPN clients. Clients initially connect to a master server and are redirected to the least-loaded server within the cluster, using the IKEv2 redirection mechanism as described in RFC5685 “Redirect Mechanism for the Internet Key Exchange Protocol Version 2 (IKEv2).”

FlexVPN Load Balancer uses the concept of a cluster of VPN servers. These servers must be in the same layer 2 network and be part of the same HSRP group. The HSRP active router will also be the master of the cluster, owning the virtual IP address (VIP) for the cluster, with other routers in the HSRP group in standby state as cluster members (also known as slaves).

All members in the cluster communicate their load to the cluster master, the cluster master maintains a record of all cluster members and their load, which is then used when calculating the least-loaded member.

When a client connects to HSRP VIP, which is the cluster master, the master will redirect the client to the physical address of the least-loaded cluster member. The client will then initiate a new IKEv2 session directly to the cluster member that it was redirected to. This behavior is illustrated in Figure 11-4.

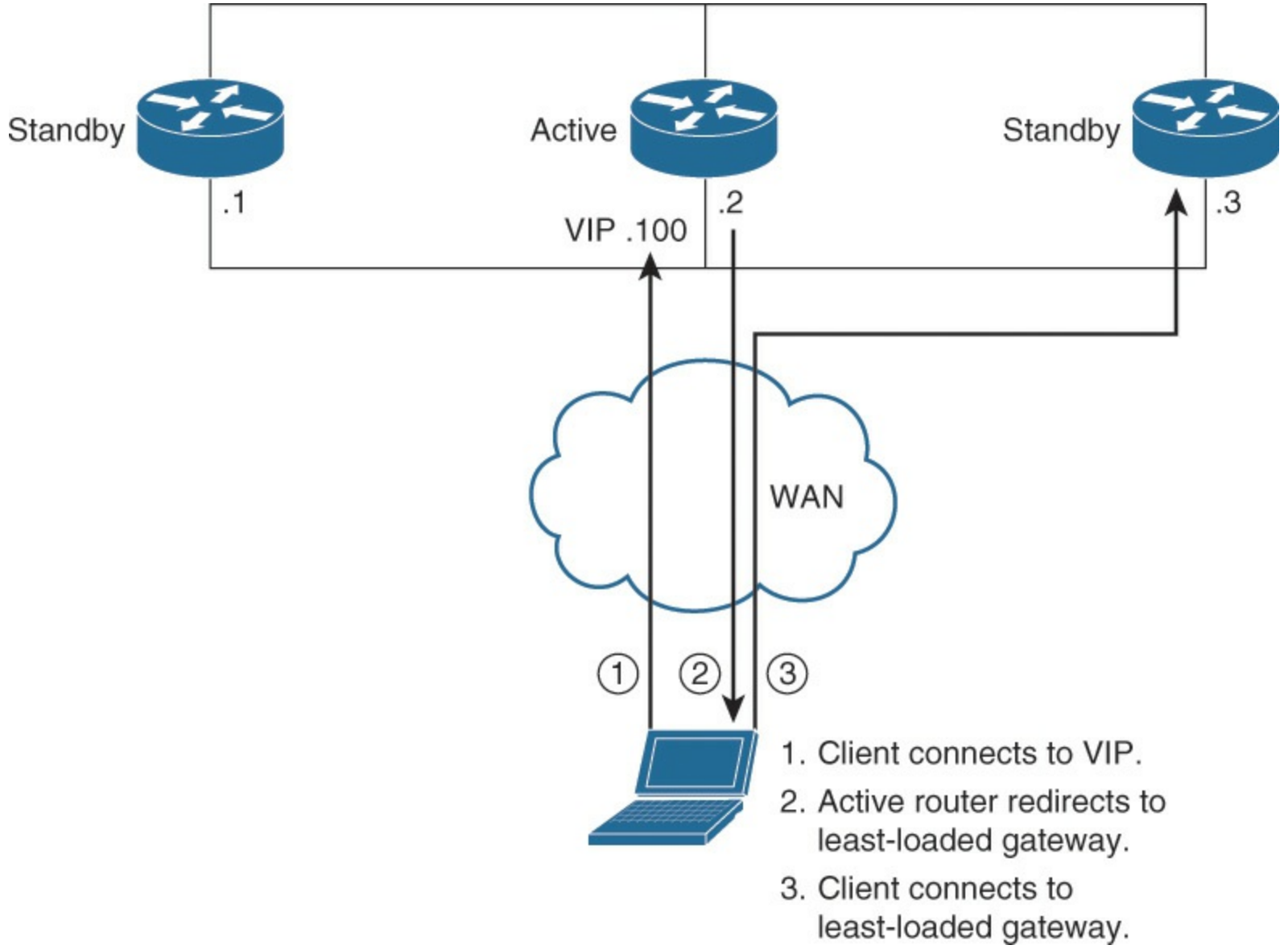


Figure 11-4 *IKEv2 Redirect*

Configuration on the client is simple when the clients connect only to the IP address or FQDN of the HSRP VIP, resulting in minimal room for configuration error. The ability to add new cluster members allows the cluster to be in a pay-as-you-grow fashion. A cluster can be built with a small number of members (or even only one), with then new hardware added as required. The hardware and software of devices do not have to match, resulting in the ability to add hardware with enough capacity to service the required additional clients and to upgrade cluster members without having to bring the whole cluster down.

Because the load is evenly distributed between members, assuming that a load is evenly distributed between cluster members and a new member is added to the cluster, the new member will receive all new requests until its load matches all other members.

To configure the IKEv2 cluster feature, the **crypto ikev2 cluster** command is used to enter the cluster load management feature. The sub-menu contains attributes that can be used to customize the cluster. The following example illustrates how to configure the feature.

Click here to view code image

```
Router(config)#crypto ikev2 cluster
Router(config-ikev2-cluster)#
```

There are three mandatory components to this feature.

- The HSRP group must be configured, using the **standby-group** *group_name* command.
- The maximum number of IPsec and IKEv2 SAs must be configured by using the **slave max-session** ***number*** command. The number is configurable between 1 and 100,000.
- Once the cluster feature has been configured, IKEv2 clustering must be enabled using the **no shutdown** command within the IKEv2 cluster. This will then activate the cluster feature.

Note

Before this command is enabled, the cluster will show as shut down:

Click here to view code image

```
Router#show crypto ikev2 cluster
Cluster is shutdown
```

The following sections describe specific configuration attributes that relate to the IKEv2 cluster feature. For this feature to be active, the cluster member must also enable the sending of the IKEv2 Redirect.

Cluster Load

The FlexVPN cluster master maintains a status of the load on all other cluster members. Cluster members connect to the master, using a Cisco propriety protocol called Cisco load balancing (CLB) protocol. CLB operates over TCP port 2012 and is used by cluster members to inform the master of their state (enabling **debug crypto ikev2 cluster detail** will display details of the load being sent from cluster members to the cluster master). Cluster members send details of their load at 3000 ms intervals; the cluster member will also send a hello packet every 1000 ms, which will be acknowledged by the master. There is a default hold time of 3000 ms. Because the hello time is set to 1000 ms, three missed hellos will be required to occur before the sessions are pulled down. This ensures that the session between the cluster and the master is not torn down should one or two hello packets be missed for reasons, such as congestion or network re-convergence.

The command **show crypto ikev2 cluster** on the cluster master will show the status of the load on cluster members. The least-loaded cluster member will be denoted with a star (*) next to it, and it will have the next IKEv2 session redirect to it.

The following example illustrates how to view the status of an IKEv2 cluster master. We see 10.10.10.3 is the least-loaded member of the cluster and will have the next IKEv2 session redirected to it.

Click here to view code image

```
Router#show crypto ikev2 cluster
Role            : CLB Master
Status          : Up
CLB Slaves      : 2
Cluster IP      : 10.10.10.100
Hold time       : 3000 msec
Overload limit : 90%
Codes           : '*' Least loaded, '-' Overloaded

Load statistics:
```

```
  Gateway           Role   Last-seen  Prio   Load     IKE  Ipsec
  ------------------------------------------------------------
 -10.10.10.2      Master      --       100   90.0%      1      2
 *10.10.10.3       Slave  00:00.728   100   13.2%      0      0
  10.10.10.1       Slave  00:00.766   100   28.2%      1      2
```

To configure the port used and the timers, adjust the following parameters under the ikev2 cluster:

holdtime *100-120000*

The hold time interval is in milliseconds. The default is 3000 ms.

port 1-65535

TCP port for CLB. The default is 2012.

slave hello *100-30000*

Slave hello time in milliseconds. The default is 1000 ms.

slave max-session *1-100000*

Maximum number of IKEv2 and IPsec Security Associations. The default is 10.

slave update *100-60000*

Load update time in milliseconds. The default is 3000 ms.

slave priority *1-100*

Slave priority is a value set on the cluster member that influences the load calculation. The higher the slave priority, the lower is the load calculated on the cluster member. The following example shows Router1 as a member of the cluster, which has an initial slave priority of 100 and a load of 13.2%. After amending the slave priority to 50 on Router1, we see the load increase to 19.8%.

Click here to view code image

```
Router1(config)#crypto ikev2 cluster
Router1(config-ikev2-cluster)#do show crypto ikev2 cluster

Role           : CLB Slave
Status         : Up
Cluster IP     : 10.10.10.100
Hold time      : 3000 msec
Hello-interval : 1000 msec
Update-interval: 3000 msec

Load statistics:
  Gateway            Last-Ack  Prio   Load     IKE  IPsec
  ------------------------------------------------------
  10.10.10.3       00:00.725   100   13.2%      0      0


Router1(config-ikev2-cluster)#slave priority 50
Router1(config-ikev2-cluster)#do show crypto ikev2 cluster

Role           : CLB Slave
Status         : Up
Cluster IP     : 10.10.10.100
Hold time      : 3000 msec
```

```
Hello-interval : 1000 msec
Update-interval: 3000 msec

Load statistics:
  Gateway            Last-Ack  Prio   Load     IKE  IPsec
  ---------------------------------------------------------
  10.10.10.3       00:00.865    50   19.8%       0      0
```

The ability to amend the load can be used to amend the configurable values to ensure that the master unit is seen as the most loaded member in the cluster. This can be useful to ensure that the master unit is not overloaded with IKEv2 sessions, so it has capacity to reply to redirect clients to other cluster members and also service any clients that do not support the redirect feature.

The standby group is used to tie the HSRP group to the cluster.

The load of cluster members is calculated using the following parameters:

- System load—Memory percentage used
- System load—Average CPU usage for past 5 seconds
- Crypto load—IPsec Security Associations
- Crypto load—IKEv2 SAs
- Crypto load—CAC limit
- Overload limit
- Slave Priority

The system and crypto load parameters can be configured to have less precedence when calculating the percentage of load on a cluster member. By default, the weight is set to 100; however, this can be lowered, which will result in the load being calculated to a lower value. This is beneficial in circumstances in which different hardware is used for cluster members.

The following example illustrates the IKEv2 cluster weight configuration.

Click here to view code image

```
crypto ikev2 cluster
 master weight crypto-load 100
 master weight system-load 100
```

The overload limit is the load limit that is calculated locally for the cluster member. Should a cluster member reach this limit, it will be marked as unavailable and will not accept any more connections. The default is set to 90%.

The following example illustrates the IKEv2 cluster overload-limit configuration.

```
crypto ikev2 cluster
 master overload-limit 90
```

IKEv2 Redirect

The ability to redirect a FlexVPN client to another VPN gateway is based on the standard described in RFC5685 Redirect Mechanism for the Internet Key Exchange Protocol Version 2 (IKEv2).

For clients to have the ability to use the IKEv2 redirect mechanism, the client must explicitly specify that it supports IKEv2 redirects. This is achieved by including a REDIRECT_SUPPORTED payload, which is sent in the initial IKE_SA_INIT message. This can be configured on the client by the

command **crypto ikev2 client redirect.** If the IKE_SA_INIT request did not indicate support for the redirect mechanism, the responder *must not* send the REDIRECT payload to the VPN client.

The cluster master can send the IKEv2 redirect in either the IKE_SA_INIT or in the IKE_AUTH exchange. Sending the IKEv2 redirect in the encrypted IKE_AUTH exchange offers the ability to redirect only authenticated clients because the redirect will occur after both parties have performed a mutual authentication.

On the cluster member, the global command to configure sending redirects is **crypto ikev2 redirect gateway init** to enable sending of redirects in IKE_SA_INIT or **crypto ikev2 redirect gateway auth** to send a redirect in IKE_AUTH.

If the ability to send IKEv2 redirects is enabled when the cluster feature is not enabled, a warning message is generated as illustrated in the following example.

Click here to view code image

```
Router(config)#crypto ikev2 redirect gateway init
 ! (IKEv2 Cluster load-balancer is not enabled)
```

Within the IKEv2 profile, the ability to send an IKEv2 redirect can be configured, using the following command:

Click here to view code image

```
crypto ikev2 profile default
 redirect gateway auth | init
```

Note that Cisco AnyConnect explicitly contains the IKEv2 redirect capability.

Once the cluster master has sent the IKEv2 redirect, the client will then initiate a new connection to the peer that was indicated in the reply from the cluster master. Within the new IKE_SA_INIT exchange a REDIRECTED_FROM payload will be included, which contains the IP address of the gateway that sent the redirect. In the case of IKEv2 load balancer, this will be the VIP. This payload also serves the purpose of indicating support for the redirect mechanism to the new VPN gateway after the client has received a redirect.

When a client is redirected to another IKEv2 peer, the initial IKEv2 SA that was used by the redirecting gateway will be deleted. This can be seen in the following example when the client receives the responder's IKE_SA_INIT containing the redirect; the original SA is then deleted.

Click here to view code image

```
IKEv2 IKE_SA_INIT Exchange RESPONSE
Payload contents:
 NOTIFY(REDIRECT)
IKEv2:(SA ID = 1):Processing IKE_SA_INIT message
IKEv2:(SA ID = 1):Redirect acceptance check with platform
IKEv2:(SA ID = 1):Redirect accepted, clean-up IKEv2 SA, platform will initiate
new request
IKEv2:(SA ID = 1):
IKEv2:(SA ID = 1):Failed SA init exchange
IKEv2:(SA ID = 1):Initial exchange failed
IKEv2:(SA ID = 1):Initial exchange failed
IKEv2:(SA ID = 1):Abort exchange
IKEv2:(SA ID = 1):Deleting SA
```

Redirect Loops

A client request could be redirected multiple times in a sequence because of either an incorrect configuration or a denial-of-service (DoS) attack. If a client were redirected continuously between gateways, the client would never establish a session with its intended peer, and resources on the gateways redirecting the client could be consumed resulting in loss of service.

To prevent the possibility of clients being redirected in loops, a client can be configured, using the **crypto ikev2 redirect client** command with the **max-redirects** keyword argument pair, to not accept more than a specific number of redirects for a particular IKEv2 SA setup.

The value that can be configured is between 1 and 255 with a default of 5. Should a client meet this limit, it will abort the connection attempt.

FlexVPN Client

For a FlexVPN client to access an IKEv2 load balancer solution, the client must be configured with the **crypto ikev2 redirect client** command that enables the client to advertise the ability to use IKEv2 redirect and also to act on it.

Additionally, the tunnel interface configured on the client must not have a static destination address. The destination address of the tunnel must be set to dynamic because the client will first connect to the VIP and then be redirected to the final destination address. If the tunnel is configured with a static destination address, the client will not act on the IKEv2 redirect.

To enable a FlexVPN client to use a dynamic address, the client should be configured to use a tunnel interface with a dynamic destination. The following example illustrates a FlexVPN client configuration with a dynamic destination address.

Click here to view code image

```
crypto ikev2 client flexvpn flex_client
  peer 1 10.10.10.1
  client connect Tunnel1

interface Tunnel1
 ip address negotiated
 tunnel source Ethernet0/0
 tunnel destination dynamic
 tunnel protection ipsec profile default
```

> **Note**
>
> The Cisco AnyConnect client supports the ability to use IKEv2 redirect and can integrate with the IKEv2 cluster.

Troubleshooting IKEv2 Load Balancing

Should you experience issues with the IKEv2 load-balancing feature the command **show crypto ikev2 cluster** will detail the role of the cluster member (Master or Slave) along with details of the load of the device. As previously mentioned, this command issued on the cluster master will display the least-loaded member, which will be the next device redirected to.

The following example illustrates the **show standby** command that will display details of the HSRP

configuration and status.

Click here to view code image

```
Router#show standby
Ethernet0/0 - Group 1
  State is Listen
  Virtual IP address is 10.10.10.100
  Active virtual MAC address is 0000.0c07.ac01
    Local virtual MAC address is 0000.0c07.ac01 (v1 default)
  Hello time 3 sec, hold time 10 sec
  Preemption disabled
  Active router is 10.10.10.2, priority 255 (expires in 9.120 sec)
  Standby router is 10.10.10.3, priority 100 (expires in 10.816 sec)
  Priority 100 (default 100)
  Group name is "group1" (cfgd)
```

Debugging can be enabled specifically for the ikev2 cluster feature by using the **debug crypto ikev2 cluster** command; this will enable specific debug information pertaining to the IKEv2 cluster such as received or sent messages.

The **debug crypto ikev2 cluster detail** command will enable detailed debugging, which will create a very verbose output, containing detailed information about messages and contents and load information calculated for all update messages received. It should be noted that as the output is quite verbose, caution should be exercised to prevent overloading of lower-end routers.

The **debug crypto ikev2 cluster error** will enable debugging for errors encountered within the IKEv2 cluster feature.

Standard IKEv2 debug commands such as **debug crypto ikev2** and **debug crypto ikev2 packet hexdump** can be used to verify the REDIRECT payloads being included within the IKEv2 exchange.

The following example illustrates when the **debug crypto ikev2 packet hexdump** command was enabled on a client. After redirection, we see the client initiate a connection including the REDIRECT_FROM payload, and we can confirm that 0A 0A 0A 64 is the HEX for 10.10.10.100 and thus can determine where the REDIRECTED_FROM payload originated from

Click here to view code image

```
NOTIFY(REDIRECTED_FROM)  Next payload: NONE, reserved: 0x0, length: 14
    Security protocol id: Unknown - 0, spi size: 0, type: REDIRECTED_FROM
01 04 0A 0A 0A 64
```

IKEv2 Load Balancer Example

In this section, we will describe a typical IKEv2 load balancer deployment. Figure 11-5 illustrates the IKEv2 load balancer in action. Three routers (Router1, Router2, and Router3) are configured as a cluster, with a number of clients connected to these.

The WAN facing HSRP configuration uses the IP address 10.10.10.100 for the VIP. Each router also has a unique IP address configured with Router1 having 10.10.10.1, Router2 having 10.10.10.2, and Router3 having 10.10.10.3.

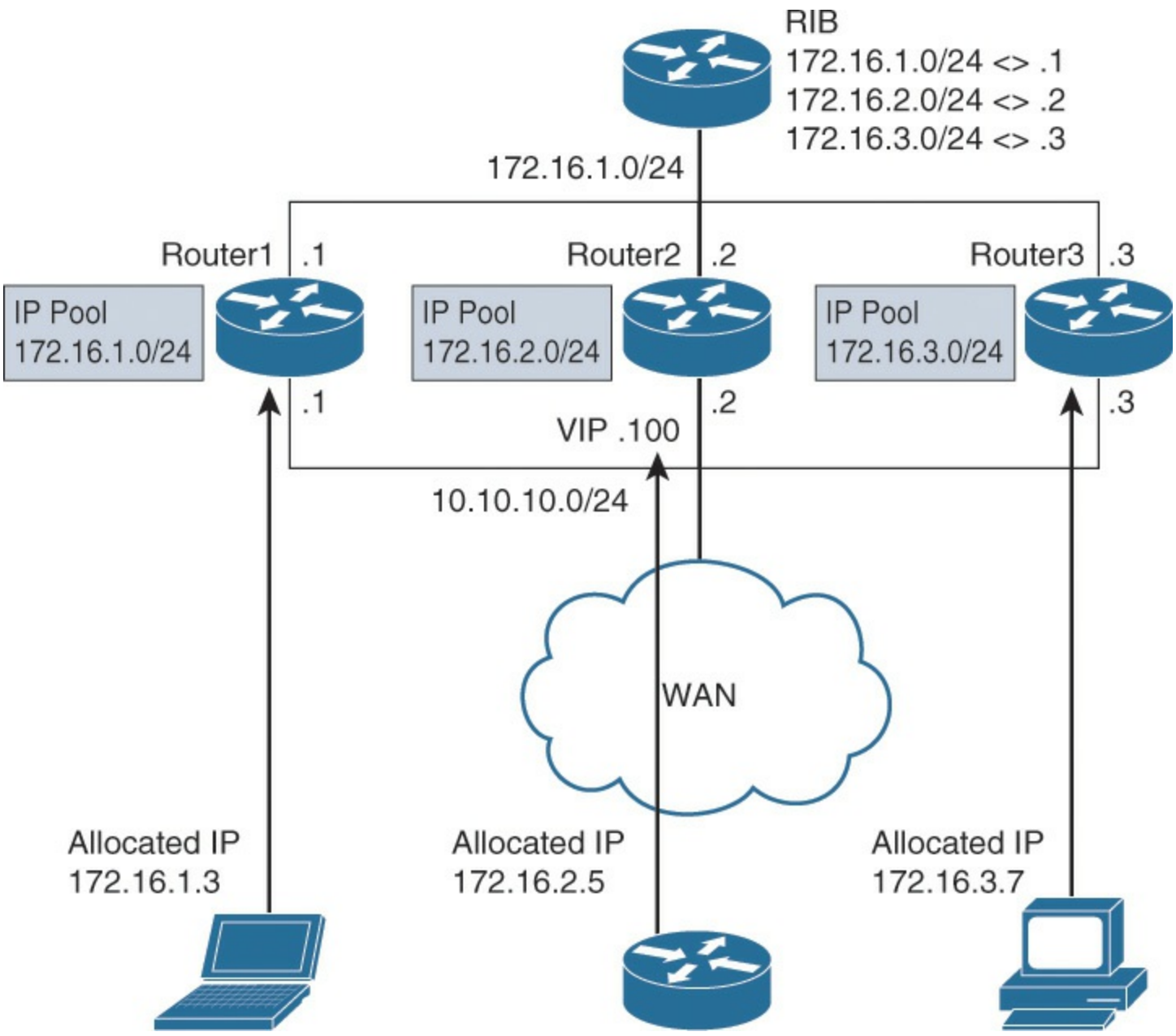


Figure 11-5 *Load Balancer Example*

The router northbound of the load balancer must be aware of the address pools that are configured to allocate clients IP addresses on the cluster. This allows for clients that access northbound data to then allow the returning traffic back to the correct cluster member, which is then encapsulated and sent back to the client.

The following configuration example from Router1 illustrates the configuration required to enable the IKEv2 cluster feature along with the necessary IKEv2 configuration to enable remote access clients to connect.

> **Note**
>
> The configuration for interfaces and the certificate authority and certificate map have been removed.

HSRP is configured on the WAN facing interface with a VIP of 10.10.10.100 and a name of "group1." For all other routers within the cluster, the IP address will be unique. The VIP and group name match what is configured in the following example.

Click here to view code image

```
interface Ethernet0/1
 ip address 10.10.10.1 255.255.255.0
 standby 1 ip 10.10.10.100
 standby 1 name group1
```

IKEv2 redirects are enabled by using the **crypto ikev2 redirect gateway init** command with clients redirected in the IKE_SA_INIT exchange.

The following example shows the IKEv2 cluster feature configuration, with the HSRP group name configured in the previous example.

```
crypto ikev2 cluster
 standby-group group1
 slave max-session 10
 no shutdown
```

The following example shows the IKEv2 configuration that is required to allow remote clients to connect. The local pool configuration is unique to this router, and other cluster members will have a different range configured with the pool. The rest of the configuration will be used across all other cluster members.

Click here to view code image

```
crypto ikev2 profile default
 match certificate certmap
 identity local dn
 authentication remote rsa-sig
 authentication local rsa-sig
 pki trustpoint 1
 aaa authorization group cert list default default
 virtual-template 1

crypto ikev2 authorization policy default
 pool pool1
 route set interface

interface Virtual-Template1 type tunnel
 ip unnumbered Loopback1
 tunnel source Ethernet0/1
 tunnel protection ipsec profile default

ip local pool pool1 172.16.1.1 172.16.1.255
```

The following example illustrates the master (Router1) that has a single client session connected to it. The next IKEv2 session to be received on VIP of the master will be redirected to Router3 (10.10.10.3).

Click here to view code image

```
Router1#show crypto ikev2 cluster

Role           : CLB Master
Status         : Up
CLB Slaves     : 2
Cluster IP     : 10.10.10.1
Hold time      : 3000 msec
Overload limit : 90%
```

```
Codes          : '*' Least loaded, '-' Overloaded

Load statistics:
  Gateway           Role   Last-seen  Prio   Load     IKE  IPsec
  ------------------------------------------------------------
  10.10.10.1       Master     --       100   31.7%      1      2
 *10.10.10.3        Slave  00:00.224   100   16.2%      0      0
  10.10.10.2        Slave  00:00.600   100   16.2%      0      0
```

The following example illustrates the syslog messages, which were generated using **debug crypto ikev2** on the master, and shows a client connecting to the VIP and being redirected to Router3. In the IKE_SA_INIT request, we can see that the REDIRECT_SUPPORTED notify payload is included.

Click here to view code image

```
IKEv2:Received Packet [From 10.10.20.6:500/To 10.10.10.1:500/VRF i0:f0]
Initiator SPI : 5DF2CD9E0A2E1171 - Responder SPI : 0000000000000000 Message id: 0
IKEv2 IKE_SA_INIT Exchange REQUEST
Payload contents:
 SA KE N VID VID NOTIFY(NAT_DETECTION_SOURCE_IP)
NOTIFY(NAT_DETECTION_DESTINATION_IP) NOTIFY(REDIRECT_SUPPORTED)
```

The master will then redirect the client to the least-loaded gateway (10.10.10.3) by sending a reply containing a redirect notify with this information.

Click here to view code image

```
IKEv2:(SESSION ID = 10,SA ID = 1):Redirect check with platform for load-balancing
IKEv2:(SA ID = 1):IKEv2 SA got Redirected to '10.10.10.3'
IKEv2:(SESSION ID = 10,SA ID = 1):Sending Redirect notification

IKEv2:(SESSION ID = 10,SA ID = 1):Sending Packet [To 10.10.20.6:500/From
10.10.10.1:500/VRF i0:f0]
Initiator SPI : 5DF2CD9E0A2E1171 - Responder SPI : 0000000000000000 Message id: 0
IKEv2 IKE_SA_INIT Exchange RESPONSE
Payload contents:
 NOTIFY(REDIRECT)
```

The client will then attempt to establish an IKEv2 SA with 10.10.10.3.

Summary

The FlexVPN load balancer feature combines HSRP and the IKEv2 redirect feature to provide a scalable IKEv2 VPN headend solution. Additional hardware can be added to the headend cluster, allowing for a pay-as-you-grow model where new capacity can be easily added when required.

Because the IKEv2 redirect feature is standardized, any compatible third-party VPN client based on IKEv2 can be used with the FlexVPN load balancer. Clients connect to the HSRP VIP and are then redirected to the least-loaded gateway. This allows for an equal load to be distributed across cluster members.

Chapter 12. FlexVPN Deployments

Introduction

This chapter covers the various FlexVPN deployment scenarios, based on the FlexVPN features explained in the previous chapters. The FlexVPN features were developed to simplify the deployments and address the scalability and the various resiliency requirements. This chapter provides sample configurations and verification steps for some of these deployment scenarios.

The chapter covers the following scenarios:

- FlexVPN AAA-based pre-shared keys
- FlexVPN user and group authorization
- FlexVPN routing, dual stack, and tunnel mode auto
- FlexVPN client NAT to server-assigned IP address
- FlexVPN WAN resiliency, using dynamic tunnel source
- FlexVPN hub resiliency, using backup peers
- FlexVPN backup tunnel, using track-based tunnel activation

FlexVPN AAA-Based Pre-Shared Keys

This scenario illustrates the scalable management of per-peer pre-shared keys using a RADIUS server. Figure 12-1 illustrates the topology with two branch offices connecting to the headquarters in a FlexVPN spoke-hub topology, using pre-shared key authentication.

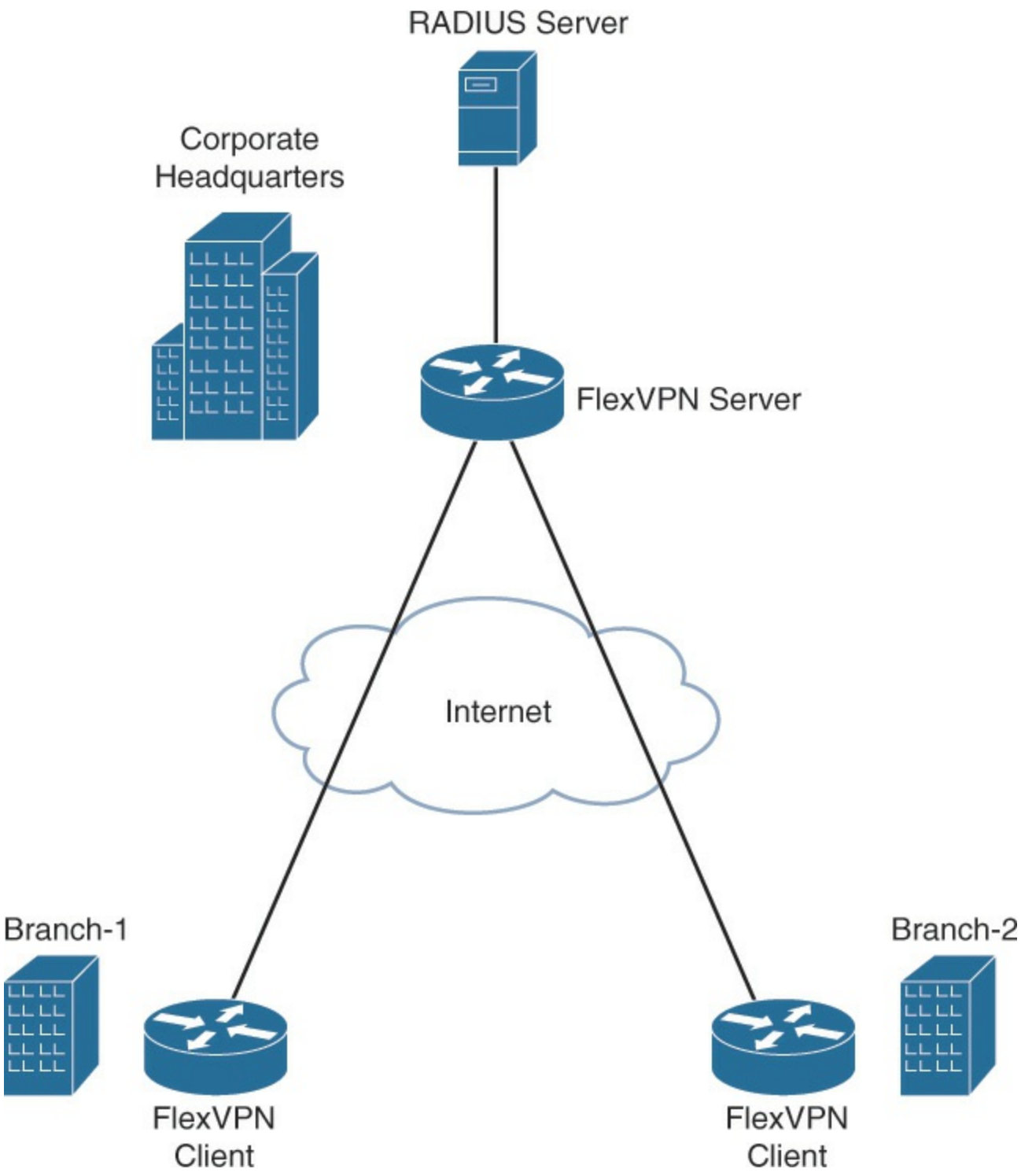


Figure 12-1 *FlexVPN AAA-Based Pre-Shared Keys Topology*

The following example illustrates the configuration and verification of the FlexVPN AAA-based pre-shared key authentication on the branch and the hub routers. To demonstrate the configuration of symmetric as well as asymmetric pre-shared keys on the RADIUS server, the branch-1 router is configured to use symmetric pre-shared keys, and the branch-2 router is configured to use asymmetric pre-shared keys. The branch routers use a local IKEv2 keyring because only the pre-shared keys for the hub router are required. The hub router uses AAA-based pre-shared keys because the hub needs to have pre-shared keys for all the branch routers.

Configuration on the Branch-1 Router

The branch-1 router is configured with a unique local IKEv2 identity and a local IKEv2 keyring with a symmetric pre-shared key for authentication with the hub router at headquarters.

Click here to view code image

```
crypto ikev2 keyring local_keyring
 peer hub-router
  address 172.16.1.2
  pre-shared-key branch1-hub-key

crypto ikev2 profile default
 match identity remote fqdn hq.example.com
 identity local fqdn branch1.example.com
 authentication local pre-share
 authentication remote pre-share
 keyring local local_keyring
```

Configuration on the Branch-2 Router

The branch-2 router is configured with a unique local IKEv2 identity and a local IKEv2 keyring with asymmetric pre-shared keys for authentication with the hub router at headquarters.

Click here to view code image

```
crypto ikev2 keyring local_keyring
 peer hub-router
  address 172.16.1.2
  pre-shared-key local branch2-to-hub-key
  pre-shared-key remote hub-to-branch2-key

crypto ikev2 profile default
 match identity remote fqdn hq.example.com
 identity local fqdn branch2.example.com
 authentication local pre-share
 authentication remote pre-share
 keyring local local_keyring
```

Configuration on the Hub Router

The AAA authorization is configured to use the RADIUS server to retrieve the IKEv2 pre-shared keys.

Click here to view code image

```
aaa new-model

aaa group server radius radius_group1
 server name radius_server1

aaa authorization network aaa_psk_list group radius_group1

radius server radius_server1
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_server1_key
```

The IKEv2 name mangler is configured to derive the AAA username from the hostname portion of the peer IKEv2 identity of type FQDN. When each branch router is configured with a unique local FQDN identity, the name mangler will yield a unique AAA username for the pre-shared key lookup on the RADIUS server.

Click here to view code image

```
crypto ikev2 name-mangler aaa_psk_name_mangler
 fqdn hostname
```

The IKEv2 profile is configured to match all the branch routers, based on the domain portion (**example.com**) of the peer FQDN identity. The profile is configured to use an AAA-based keyring that would retrieve the pre-shared keys, using AAA authorization from the RADIUS server specified in the referenced AAA method list. The referenced IKEv2 name mangler will yield a unique AAA username for pre-shared key lookup on the RADIUS server that is derived from the username portion the peer FQDN identity.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote fqdn domain example.com
 identity local fqdn hq.example.com
 authentication local pre-share
 authentication remote pre-share
 keyring aaa aaa_psk_list name-mangler aaa_psk_name_mangler
```

Configuration on the RADIUS Server

The RADIUS server is configured with a profile for each branch router. Note that the following configuration illustrates the Cisco VSA attribute format. The exact configuration steps would vary with the RADIUS implementation.

RADIUS profile for username “branch1”:

Click here to view code image

```
 vsa cisco generic 1 string "ipsec:tunnel-password=branch1-key"
```

RADIUS profile for username “branch2”:

Click here to view code image

```
vsa cisco generic 1 string "ipsec:ikev2-password-local=hub-to-branch2-key"
vsa cisco generic 1 string "ipsec:ikev2-password-remote=branch2-to-hub-key"
```

The following example illustrates the logs on the hub capturing the retrieval of the symmetric pre-shared key from the RADIUS server for the branch-1 router. The **debug crypto ikev2**, **debug crypto ikev2 internal**, and **debug radius authentication** are enabled. Due to the verbose nature, internal debugging should be enabled only under the guidance of Cisco TAC.

Click here to view code image

```
IKEv2-INTERNAL:Fetching shared secret from AAA
IKEv2-INTERNAL:Name-mangler 'aaa_psk_name_mangler' returning mangled-name
'branch1' for name-type 1, name-len 19, name_string branch1.example.com
IKEv2-INTERNAL:Using AAA method list aaa_psk_list for peer branch1
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Password request sent

RADIUS(0000019D): Send Access-Reques t to 172.16.1.3:1645 id 1645/145, len 133
RADIUS:  authenticator 15 BF 00 82 37 B0 1B 60 - C2 55 00 9F 17 23 A9 5B
RADIUS:  User-Name           [1]   9   "branch1"
RADIUS:  User-Password       [2]   18  *
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  62
RADIUS:   Cisco AVpair       [1]   56  "audit-session-id=L2L4AC1G102ZO2L4AC1G101Z
I1F401F4ZM195"
```

```
RADIUS:  Service-Type        [6]   6   Outbound                 [5]
RADIUS:  NAS-IP-Address      [4]   6   172.16.1.2
RADIUS(0000019D): Sending a IPv4 Radius Packet
RADIUS(0000019D): Started 5 sec timeout

RADIUS: Received from id 1645/145 172.16.1.3:1645, Access-Accept, len 61
RADIUS:  authenticator E0 55 BA 88 22 3E FC E6 - AF DA 03 B3 22 8C 76 8E
RADIUS:  Vendor, Cisco       [26]  41
RADIUS:   Cisco AVpair       [1]   35  "ipsec:tunnel-password=branch1-key"
RADIUS(0000019D): Received from id 1645/145
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received password response
IKEv2-INTERNAL:Received symmmetric password from radius server
IKEv2:[Crypto Engine -> IKEv2] IKEv2 authentication data generation PASSED
IKEv2:(SESSION ID = 405,SA ID = 1):Verification of peer's authenctication data
PASSED
```

The following example illustrates the logs on the hub capturing the retrieval of the asymmetric pre-shared keys from the RADIUS server for the branch-2 router. The **debug crypto ikev2**, **debug crypto ikev2 internal**, and **debug radius authentication** are enabled.

Click here to view code image

```
IKEv2-INTERNAL:Fetching shared secret from AAA
IKEv2-INTERNAL:Name-mangler 'aaa_psk_name_mangler' returning mangled-name
'branch2' for name-type 1, name-len 19, name_string branch2.example.com
IKEv2-INTERNAL:Using AAA method list aaa_psk_list for peer branch2
IKEv2:(SA ID = 2):[IKEv2 -> AAA] Password request sent

RADIUS(0000019F): Send Access-Request to 172.16.1.3:1645 id 1645/147, len 133
RADIUS:  authenticator E9 CB 55 31 65 06 83 48 - 00 60 15 3E 87 93 3C A9
RADIUS:  User-Name           [1]   9   "branch2"
RADIUS:  User-Password       [2]   18  *
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.4"
RADIUS:  Vendor, Cisco       [26]  62
RADIUS:   Cisco AVpair       [1]   56  "audit-session-
id=L2L4AC1G102ZO2L4AC1G104ZI1F401F4ZM197"
RADIUS:  Service-Type        [6]   6   Outbound                 [5]
RADIUS:  NAS-IP-Address      [4]   6   172.16.1.2
RADIUS(0000019F): Sending a IPv4 Radius Packet
RADIUS(0000019F): Started 5 sec timeout

RADIUS: Received from id 1645/147 172.16.1.3:1645, Access-Accept, len 127
RADIUS:  authenticator F3 20 73 8E A8 B9 9F B8 - F0 02 A4 CA 64 47 66 82
RADIUS:  Vendor, Cisco       [26]  53
RADIUS:   Cisco AVpair       [1]   47  "ipsec:ikev2-password-local=hub-to-
branch2-key"
RADIUS:  Vendor, Cisco       [26]  54
RADIUS:   Cisco AVpair       [1]   48  "ipsec:ikev2-password-remote=branch2-to-
hub-key"
IKEv2:(SA ID = 2):[AAA -> IKEv2] Received password response
IKEv2-INTERNAL:Received local password from radius server
IKEv2-INTERNAL:Received remote password from radius server
IKEv2:[Crypto Engine -> IKEv2] IKEv2 authentication data generation PASSED
IKEv2:(SESSION ID = 407,SA ID = 2):Verification of peer's authenctication data
PASSED
```

FlexVPN User and Group Authorization

This FlexVPN remote access scenario demonstrates the use of FlexVPN group and user authorization to enforce common and specific policy for FlexVPN clients using group and user authorization on the FlexVPN server. The scenario uses two branch routers with the FlexVPN client connected to the FlexVPN server at the headquarters as illustrated in Figure 12-1. Each of the FlexVPN clients is configured to use a unique local IKEv2 identity of type FQDN. The FlexVPN server uses group authorization to retrieve the common enterprise DNS server IP address to be pushed to all clients and user authorization to retrieve a unique IP address to be assigned to each client.

The following example illustrates the FlexVPN client configuration at the branch routers and the FlexVPN server configuration at the headquarters.

FlexVPN Client Configuration at Branch 1

The FlexVPN client is configured with a unique local IKEv2 identity.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote fqdn hq.example.com
 identity local fqdn branch1.example.com
 authentication local pre-share
 authentication remote pre-share
 keyring local local_keyring

crypto ikev2 client flexvpn flex_client_profile
  peer 1 172.16.1.2
  connect manual
  client connect Tunnel0

interface Tunnel0
 ip address negotiated
 tunnel source Ethernet0/0
 tunnel destination dynamic
 tunnel protection ipsec profile default
```

FlexVPN Client Configuration at Branch 2

The FlexVPN client is configured with a unique local IKEv2 identity.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote fqdn hq.example.com
 identity local fqdn branch2.example.com
 authentication local pre-share
 authentication remote pre-share
 keyring local local_keyring

crypto ikev2 client flexvpn flex_client_profile
  peer 1 172.16.1.2
  connect manual
  client connect Tunnel0

interface Tunnel0
 ip address negotiated
 tunnel source Ethernet0/0
```

```
tunnel destination dynamic
tunnel protection ipsec profile default
```

Configuration on the FlexVPN Server

The AAA authorization is configured to use the local AAA and the RADIUS server for FlexVPN group and user authorizations.

Click here to view code image

```
aaa new-model

aaa group server radius radius_group1
 server name radius_server1

aaa authorization network user_author_list group radius_group1
aaa authorization network group_author_list local

radius server radius_server1
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_server1_key
```

The IKEv2 name manglers are configured to derive the AAA username for the user and group authorizations from the hostname and the domain portions of the peer IKEv2 identity of type FQDN.

Click here to view code image

```
crypto ikev2 name-mangler user_author_name_mangler
 fqdn hostname

crypto ikev2 name-mangler group_author_name_mangler
 fqdn domain
```

The IKEv2 authorization policy, which acts as local AAA database for FlexVPN authorizations, is used for group authorization and is configured with a name that matches the domain portion of the peer IKEv2 identity. The authorization policy defines the enterprise DNS server IP address that is pushed to FlexVPN clients at all branches.

Click here to view code image

```
crypto ikev2 authorization policy example.com
 dns 192.168.10.10
```

The IKEv2 profile is configured to match all the branch routers based on the domain portion (**example.com**) of the peer FQDN identity. The profile is configured with group authorization using local AAA as specified in the referenced AAA method list, and using the domain portion of the peer IKE identity as AAA username as specified in the referenced name mangler. The profile is configured with user authorization, using the RADIUS server as specified in the referenced AAA method list and using the hostname portion of the peer IKE identity as the AAA username as specified in the referenced name mangler. The group authorization provides the common DNS server IP address for all branches, and the user authorization provides a unique IP address for the FlexVPN client at every branch.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote fqdn domain example.com
 identity local fqdn hq.example.com
```

```
 authentication local pre-share
 authentication remote pre-share
 keyring aaa aaa_psk_list name-mangler aaa_psk_name_mangler
 aaa authorization group psk list group_author_list name-mangler group_author_
name_mangler
 aaa authorization user psk list user_author_list name-mangler user_author_name_
mangler
 virtual-template 1
```

Configuration on the RADIUS Server

The RADIUS server is configured with a profile for each FlexVPN client with the name matching the hostname portion of the client's IKE identity for user authorization. Each of the profiles is configured with a unique IP address that is assigned by the FlexVPN server to that client.

RADIUS profile for username "branch1":

Click here to view code image

```
framed address 10.0.0.1 255.255.255.0
```

RADIUS profile for username "branch2":

Click here to view code image

```
framed address 10.0.0.2 255.255.255.0
```

The following example illustrates the logs on the FlexVPN server that capture the local group authorization and RADIUS-based user authorization. The **debug crypto ikev2**, **debug crypto ikev2 internal**, and **debug radius authentication** are enabled.

Logs Specific to FlexVPN Client-1

The following logs capture local group authorization.

Click here to view code image

```
IKEv2:Using mlist group_author_list and username example.com for group author
request
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorisation request sent
IKEv2-INTERNAL:IKEv2 local AAA author request for 'example.com'
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response
IKEv2-INTERNAL:Received group author attributes:
ipv4-dns: 192.168.10.10, route-accept any tag:1 distance:1,
```

The following logs capture RADIUS-based user authorization.

Click here to view code image

```
IKEv2:Using mlist user_author_list and username branch1 for user author request
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorisation request sent
RADIUS(0000001D): Send Access-Request to 172.16.1.3:1645 id 1645/10, len 131
RADIUS:  authenticator 76 2E F4 8B 41 A0 49 05 - E4 29 56 6A D3 6D 23 97
RADIUS:  User-Name            [1]   9   "branch1"
RADIUS:  User-Password        [2]   18  *
RADIUS:  Calling-Station-Id   [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco        [26]  60
RADIUS:   Cisco AVpair        [1]   54  "audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZO6"
RADIUS:  Service-Type         [6]   6   Outbound                 [5]
RADIUS:  NAS-IP-Address       [4]   6   172.16.1.2
```

```
RADIUS(0000001D): Sending a IPv4 Radius Packet
RADIUS(0000001D): Started 5 sec timeout
RADIUS: Received from id 1645/10 172.16.1.3:1645, Access-Accept, len 73
RADIUS:  authenticator D5 CE AC 26 AA 67 89 A6 - 35 BC 09 FC 08 6B E9 F6
RADIUS:  Framed-IP-Address   [8]   6   10.0.0.1
RADIUS:  Framed-IP-Netmask   [9]   6   255.255.255.0
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response
IKEv2-INTERNAL:Received user author attributes:
ipv4-addr: 10.0.0.1, ipv4-netmask: 255.255.255.0,
```

The following logs capture FlexVPN server pushing the common DNS server and a unique IP address to the client.

Click here to view code image

```
IKEv2:(SESSION ID = 6,SA ID = 1):Config-type: Config-reply
IKEv2:(SESSION ID = 6,SA ID = 1):Attrib type: ipv4-addr, length: 4, data: 10.0.0.1
IKEv2:(SESSION ID = 6,SA ID = 1):Attrib type: ipv4-netmask, length: 4, data:
255.255.255.0
IKEv2:(SESSION ID = 6,SA ID = 1):Attrib type: ipv4-dns, length: 4, data:
192.168.10.10
```

The **show crypto ikev2 sa detailed** command on FlexVPN client-1 displays the DNS server and assigned IP address pushed by the FlexVPN server.

Click here to view code image

```
FlexClient#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA
Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         172.16.1.1/500        172.16.1.2/500        none/none            READY
      Encr: AES-CBC, keysize: 256, PRF: SHA512, Hash: SHA512, DH Grp:5, Auth sign:
PSK, Auth verify: PSK
      Life/Active Time: 86400/29 sec
      CE id: 1004, Session-id: 4
      Status Description: Negotiation done
      Local spi: 7BFDBB152E002029       Remote spi: E8700C8FDC94AA9A
      Local id: branch1.example.com
      Remote id: hq.example.com
      Pushed IP address: 10.0.0.1
      DNS Primary: 192.168.10.10
```

Logs Specific to FlexVPN Client-2

The following logs capture local group authorization.

Click here to view code image

```
IKEv2:Using mlist group_author_list and username example.com for group author
request
IKEv2:(SA ID = 2):[IKEv2 -> AAA] Authorisation request sent
IKEv2-INTERNAL:IKEv2 local AAA author request for 'example.com'
IKEv2:(SA ID = 2):[AAA -> IKEv2] Received AAA authorisation response
IKEv2-INTERNAL:Received group author attributes:
ipv4-dns: 192.168.10.10, route-accept any tag:1 distance:1,
```

The following logs capture RADIUS-based user authorization.

Click here to view code image

```
IKEv2:Using mlist user_author_list and username branch2 for user author request
IKEv2:(SA ID = 2):[IKEv2 -> AAA] Authorization request sent
```

```
RADIUS(00000021): Send Access-Request to 172.16.1.3:1645 id 1645/12, len 131
RADIUS:  authenticator CB BD 72 FD 0E 06 B4 DA - 29 4F 1D 59 56 52 25 0E
RADIUS:  User-Name             [1]   9   "branch2"
RADIUS:  User-Password         [2]   18  *
RADIUS:  Calling-Station-Id    [31]  12  "172.16.1.4"
RADIUS:  Vendor, Cisco         [26]  60
RADIUS:   Cisco AVpair         [1]   54  "audit-session-
id=L2L4AC1G102ZO2L4AC1G104ZI1F401F4ZO7"
RADIUS:  Service-Type          [6]   6   Outbound                  [5]
RADIUS:  NAS-IP-Address        [4]   6   172.16.1.2
RADIUS(00000021): Sending a IPv4 Radius Packet
RADIUS(00000021): Started 5 sec timeout
RADIUS: Received from id 1645/12 172.16.1.3:1645, Access-Accept, len 139
RADIUS:  authenticator 8D 16 2E 75 1C E8 9D 67 - CA E1 F1 E4 33 47 79 71
RADIUS:  Framed-IP-Address     [8]   6   10.0.0.2
RADIUS:  Framed-IP-Netmask     [9]   6   255.255.255.0
RADIUS(00000021): Received from id 1645/12
IKEv2:(SA ID = 2):[AAA -> IKEv2] Received AAA authorization response
IKEv2-INTERNAL:Received user author attributes:
ipv4-addr: 10.0.0.2, ipv4-netmask: 255.255.255.0,
```

The following logs capture FlexVPN server pushing the common DNS server and a unique IP address to the client.

Click here to view code image

```
IKEv2:(SESSION ID = 7,SA ID = 2):Config-type: Config-reply
IKEv2:(SESSION ID = 7,SA ID = 2):Attrib type: ipv4-addr, length: 4, data: 10.0.0.2
IKEv2:(SESSION ID = 7,SA ID = 2):Attrib type: ipv4-netmask, length: 4, data:
255.255.255.0
IKEv2:(SESSION ID = 7,SA ID = 2):Attrib type: ipv4-dns, length: 4, data:
192.168.10.10
```

The **show crypto ikev2 sa detailed** command on FlexVPN client-2 displays the DNS server and assigned IP address pushed by the FlexVPN server.

Click here to view code image

```
FlexClient2#show crypto ikev2 sa detailed
IPv4 Crypto IKEv2  SA
Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         172.16.1.4/500        172.16.1.2/500        none/none            READY
      Encr: AES-CBC, keysize: 256, PRF: SHA512, Hash: SHA512, DH Grp:5, Auth sign:
PSK, Auth verify: PSK
      Life/Active Time: 86400/7 sec
      CE id: 1003, Session-id: 3
      Status Description: Negotiation done
      Local spi: 93E45A454B87A9D5       Remote spi: 841BBA759B822DBF
      Local id: branch2.example.com
      Remote id: hq.example.com
      Pushed IP address: 10.0.0.2
      DNS Primary: 192.168.10.10
```

FlexVPN Routing, Dual Stack, and Tunnel Mode Auto

This scenario demonstrates FlexVPN routing, dual stack, and tunnel mode auto using FlexVPN spoke-hub topology to connect branch sites to the headquarters. Figure 12-2 illustrates the topology where the branch sites connect to the HQ over IPv4 and IPv6 transport.

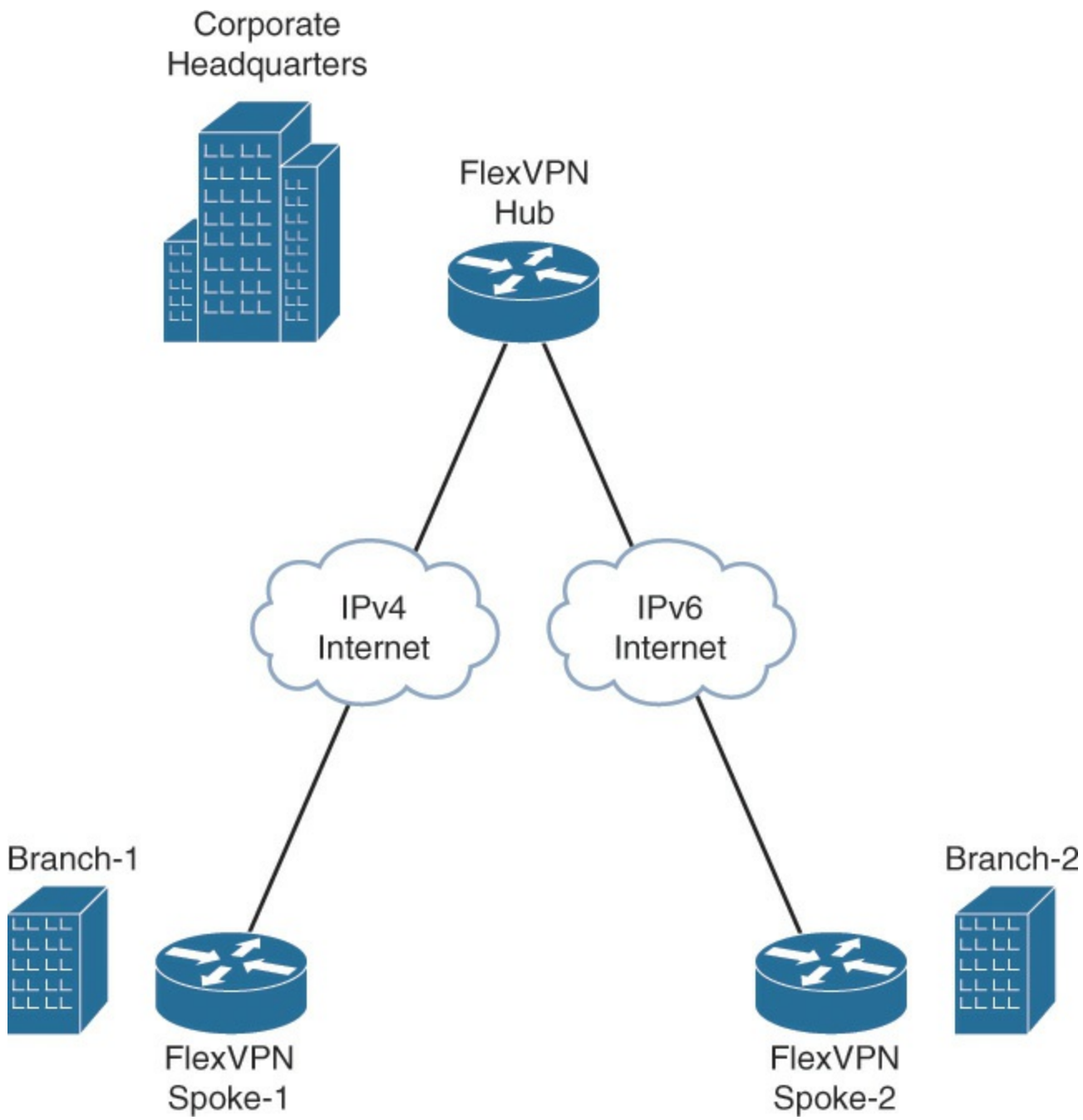


Figure 12-2 *Branch Sites Connecting to the HQ over IPv4 and IPv6*

To demonstrate FlexVPN dual stack, all the sites have IPv4 and IPv6 LAN subnets, and branch-1 is connected to the headquarters over IPv4 transport and branch-2 over IPv6 transport.

To demonstrate tunnel mode auto, the hub site uses a single virtual-template interface with an IKEv2 profile configured with tunnel mode auto to clone virtual-access interfaces for spoke1 and spoke2 that are connected over IPv4 and IPv6 transport.

The FlexVPN routing is accomplished by specifying the local IPv4 and IPv6 LAN subnets as part of the user or group authorization policy, which can be either local or on the RADIUS server. These subnets are pushed to the peer using IKEv2 configuration exchange. The IKEv2 authorization policy configurations at the spokes and the hub demonstrate the different ways of specifying the local subnets to be pushed to the peer.

The following example illustrates the FlexVPN spoke configuration at the branch routers and the FlexVPN hub configuration at the headquarters.

FlexVPN Spoke Configuration at Branch-1

The AAA authorization method list for group authorization is configured to use local AAA.

Click here to view code image

```
aaa new-model

aaa authorization network group_author_list local
```

The name mangler is configured to return the domain portion of the peer FQDN identity.

Click here to view code image

```
crypto ikev2 name-mangler group_author_name_mangler
 fqdn domain
```

Under the IKEv2 authorization policy, the **route set interface** command pushes the IPv4 and IPv6 addresses configured on the specified interface, or the virtual-access interface associated with the client VPN session if the interface is not specified, to the peer using IKEv2 configuration exchange. The peer programmatically adds static routes to these subnets pointing to the VPN tunnel interface.

Click here to view code image

```
crypto ikev2 authorization policy example.com
 route set interface Ethernet1/0
 route set interface
```

The IKEv2 profile is configured for local group authorization and the IKEv2 authorization policy acts as the local AAA database.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote fqdn hq.example.com
 identity local fqdn branch1.example.com
 authentication local pre-share key cisco
 authentication remote pre-share key cisco
 aaa authorization group psk list group_author_list name-mangler group_author_
name_mangler
```

The FlexVPN dual stack functionality is accomplished using GRE encapsulation on the tunnel interface and enabling both IPv4 and IPv6 on the tunnel interface. The **tunnel mode** command specifies the tunnel encapsulation mode as well as the transport type (IPv4 or IPv6). The default tunnel encapsulation mode is GRE-over-IPv4 (**tunnel mode gre ip**) and hence is not displayed in the running configuration. The tunnel source and destination use IPv4 addresses matching the transport type specified in **tunnel mode** command.

Click here to view code image

```
interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 ipv6 address FEC0::10:1/120
 tunnel source Ethernet0/0
 tunnel destination 172.16.1.2
 tunnel protection ipsec profile default
```

The WAN interface is IPv4 and connects to FlexVPN hub at the headquarters.

Click here to view code image

```
interface Ethernet0/0
 description IPv4 WAN interface
 ip address 172.16.1.1 255.255.255.0
```

The LAN interface has IPv4 and IPv6 addresses to simulate the IPv4 and IPv6 LAN subnets.

Click here to view code image

```
interface Ethernet1/0
 description LAN interface
 ip address 192.168.0.1 255.255.255.0
 ipv6 address FEC0::1:1/120
```

FlexVPN Spoke Configuration at Branch-2

The AAA authorization method for group authorization is configured to use local AAA.

Click here to view code image

```
aaa new-model

aaa authorization network group_author_list local
```

The name mangler is configured to return the domain portion of the peer FQDN identity.

Click here to view code image

```
crypto ikev2 name-mangler group_author_name_mangler
 fqdn domain
```

Under the IKEv2 authorization policy, along with the **route set interface** command that pushes the IPv4 and IPv6 addresses configured on the virtual-access interface associated with the client VPN session, the **route set remote ipv4|ipv6** command specifies the IPv4 and IPv6 subnets that are pushed to the peer using IKEv2 configuration exchange and the peer programmatically adds static routes to these subnets pointing to the VPN tunnel interface.

Click here to view code image

```
crypto ikev2 authorization policy example.com
 route set interface
 route set remote ipv4 192.168.1.0 255.255.255.0
 route set remote ipv6 FEC0::2:1/120
```

The IKEv2 profile is configured for local group authorization, and the IKEv2 authorization policy acts as the local AAA database.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote fqdn hq.example.com
 identity local fqdn branch2.example.com
 authentication local pre-share key cisco
 authentication remote pre-share key cisco
 aaa authorization group psk list group_author_list name-mangler group_author_
name_mangler
```

The FlexVPN dual stack functionality is accomplished using GRE encapsulation on the tunnel interface and enabling both IPv4 and IPv6 on the tunnel interface. The **tunnel mode gre ipv6** command specifies the tunnel encapsulation mode as GRE as well as the transport as IPv6. The tunnel

source and destination use IPv6 addresses matching the transport type specified in **tunnel mode** command.

Click here to view code image

```
interface Tunnel0
 ip address 10.0.0.2 255.255.255.0
 ipv6 address FEC0::10:2/120
 tunnel source Ethernet0/0
 tunnel mode gre ipv6
 tunnel destination 2001::1:2
 tunnel protection ipsec profile default
```

The WAN interface is IPv6 and connects to FlexVPN hub at the headquarters.

Click here to view code image

```
interface Ethernet0/0
 description IPv6 WAN interface
 no ip address
 ipv6 address 2001::1:1/120
```

The LAN interface has IPv4 and IPv6 addresses to simulate the IPv4 and IPv6 LAN subnets.

Click here to view code image

```
interface Ethernet1/0
 description LAN interface
 ip address 192.168.1.1 255.255.255.0
 ipv6 address FEC0::2:1/120
```

FlexVPN Hub Configuration at the HQ

The AAA authorization method for group authorization is configured to use local AAA.

Click here to view code image

```
aaa new-model

aaa authorization network group_author_list local
```

The name mangler is configured to return the domain portion of the peer FQDN identity.

Click here to view code image

```
crypto ikev2 name-mangler group_author_name_mangler
 fqdn domain
```

Under the IKEv2 authorization policy, along with the **route set interface** command that pushes the IPv4 and IPv6 addresses configured on the virtual-access interface associated with the client VPN session, the **route set access-list [ipv6]** command references the access lists, specifying the IPv4 and IPv6 subnets that are pushed to the peer by using the IKEv2 configuration exchange. The peer programmatically adds static routes to these subnets pointing to the VPN tunnel interface.

Click here to view code image

```
crypto ikev2 authorization policy example.com
 route set interface
 route set access-list v4-subnets
 route set access-list ipv6 v6-subnets
```

```
ip access-list standard v4-subnets
 permit 192.168.2.0 0.0.0.255

ipv6 access-list v6-subnets
 permit ipv6 FEC0::3:0/120 any
```

The IKEv2 profile is configured for local group authorization, and the IKEv2 authorization policy acts as the local AAA database. The **virtual-template 1 mode auto** command specifies the virtual-template used to clone the virtual-access interfaces for the spoke sessions and also enables the auto detection of tunnel encapsulation mode and the transport type from the incoming session. Accordingly, it sets the **tunnel mode** command on the cloned virtual-access interface, overriding the setting derived from the virtual-template.

Click here to view code image

```
crypto ikev2 profile default
 match identity remote fqdn domain example.com
 identity local fqdn hq.example.com
 authentication local pre-share key cisco
 authentication remote pre-share key cisco
 aaa authorization group psk list group_author_list name-mangler group_author_
name_mangler
 virtual-template 1 mode auto
```

The FlexVPN dual stack functionality is accomplished by enabling both IPv4 and IPv6 on the virtual-template interface using **ip unnumbered** and **ipv6 unnumbered** commands. Because tunnel mode auto is enabled in the IKEv2 profile, the **tunnel mode** command on the cloned virtual-access interface is derived from the incoming session. The GRE encapsulation proposed by the spokes would enable carrying both IPv4 and IPv6 traffic over the VPN tunnel.

Click here to view code image

```
interface Loopback1
 ip address 10.0.0.3 255.255.255.0
 ipv6 address FEC0::10:3/120

interface Virtual-Template1 type tunnel
 ip unnumbered Loopback1
 ipv6 unnumbered Loopback1
 tunnel protection ipsec profile default
```

The WAN interface is enabled for both IPv4 and IPv6 and connects to the FlexVPN spokes at the branches.

Click here to view code image

```
interface Ethernet0/0
 description IPv4 and IPv6 WAN interface
 ip address 172.16.1.2 255.255.255.0
 ipv6 address 2001::1:2/120
```

The LAN interface has IPv4 and IPv6 addresses to simulate the IPv4 and IPv6 LAN subnets.

Click here to view code image

```
interface Ethernet1/0
 description LAN interface
 ip address 192.168.2.1 255.255.255.0
```

```
ipv6 address FEC0::3:1/120
```

Verification on FlexVPN Spoke at Branch-1

Before the FlexVPN session is established, there are no static routes through the VPN tunnel interface

Click here to view code image

```
FlexSpoke1#show crypto session
FlexSpoke1#show ip route static
FlexSpoke1#
```

The FlexVPN session between the FlexVPN spoke1 and the hub uses GRE encapsulation over IPv4 transport.

Click here to view code image

```
FlexSpoke1#show crypto session
Crypto session current status
Interface: Tunnel0
Profile: default
Session status: UP-ACTIVE
Peer: 172.16.1.2 port 500
  Session ID: 1
  IKEv2 SA: local 172.16.1.1/500 remote 172.16.1.2/500 Active
  IPSEC FLOW: permit 47 host 172.16.1.1 host 172.16.1.2
        Active SAs: 2, origin: crypto-map
```

The **show crypto ikev2 sa detailed** command output displays the IPv4 and IPv6 subnets received from the FlexVPN hub, using IKEv2 configuration exchange. The FlexVPN spoke programmatically adds static routes through the VPN tunnel interface to the subnets received from the hub.

Click here to view code image

```
FlexSpoke1#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA
Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         172.16.1.1/500        172.16.1.2/500        none/none            READY
      Remote subnets:
      10.0.0.3 255.255.255.255
      192.168.2.0 255.255.255.0
      IPv6 Remote subnets:
      FEC0::10:3/128
      FEC0::3:0/120

FlexSpoke1#show ip route static
      10.0.0.0/8 is variably subnetted, 3 subnets, 2 masks
S        10.0.0.3/32 is directly connected, Tunnel0
S     192.168.2.0/24 is directly connected, Tunnel0

FlexSpoke1#show ipv6 route static
S   FEC0::3:0/120 [1/0], tag 1
     via Tunnel0, directly connected
S   FEC0::10:3/128 [1/0], tag 1
     via Tunnel0, directly connected
```

The FlexVPN GRE/IPv4 tunnel carries both IPv4 and IPv6 traffic to the FlexVPN hub LAN subnets learned through FlexVPN routing.

Click here to view code image

```
FlexSpoke1#clear crypto sa counters
FlexSpoke1#show crypto session detail | begin IPSEC
  IPSEC FLOW: permit 47 host 172.16.1.1 host 172.16.1.2
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 0 drop 0 life (KB/Sec) 4248161/2860
        Outbound: #pkts enc'ed 0 drop 0 life (KB/Sec) 4248160/2860

FlexSpoke1#ping ipv6 FEC0::3:1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to FEC0::3:1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexSpoke1#show crypto session detail | begin IPSEC
  IPSEC FLOW: permit 47 host 172.16.1.1 host 172.16.1.2
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 5 drop 0 life (KB/Sec) 4248160/2841
        Outbound: #pkts enc'ed 5 drop 0 life (KB/Sec) 4248159/2841

FlexSpoke1#ping 192.168.2.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.2.1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexSpoke1#show crypto session detail | begin IPSEC
  IPSEC FLOW: permit 47 host 172.16.1.1 host 172.16.1.2
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 10 drop 0 life (KB/Sec) 4248158/2805
        Outbound: #pkts enc'ed 10 drop 0 life (KB/Sec) 4248158/2805
```

Verification on FlexVPN Spoke at Branch-2

Before the FlexVPN session is established, there are no static routes through the VPN tunnel interface

Click here to view code image

```
FlexSpoke2#show crypto session
FlexSpoke2#show ip route static
FlexSpoke2#
```

The FlexVPN session between the FlexVPN spoke2 and the hub uses GRE encapsulation over IPv6 transport.

Click here to view code image

```
FlexSpoke2#show crypto session
Crypto session current status
Interface: Tunnel0
Profile: default
Session status: UP-ACTIVE
Peer: 2001::1:2 port 500
  Session ID: 1
  IKEv2 SA: local 2001::1:1/500
          remote 2001::1:2/500 Active
  IPSEC FLOW: permit 47 host 2001::1:1 host 2001::1:2
        Active SAs: 2, origin: crypto-map
```

The **show crypto ikev2 sa detailed** command output displays the IPv4 and IPv6 subnets received

from the FlexVPN hub, using IKEv2 configuration exchange. The FlexVPN spoke programmatically adds static routes through the VPN tunnel interface to the subnets received from the hub.

Click here to view code image

```
FlexSpoke2#show crypto ikev2 sa detailed
IPv6 Crypto IKEv2  SA
Tunnel-id    fvrf/ivrf              Status
1          none/none             READY
Local  2001::1:1/500
Remote  2001::1:2/500

      Remote subnets:
      10.0.0.3 255.255.255.255
      192.168.2.0 255.255.255.0
      IPv6 Remote subnets:
      FEC0::10:3/128
      FEC0::3:0/120

FlexSpoke2#show ip route static
      10.0.0.0/8 is variably subnetted, 3 subnets, 2 masks
S        10.0.0.3/32 is directly connected, Tunnel0
S     192.168.2.0/24 is directly connected, Tunnel0

FlexSpoke2#show ipv6 route static
S   FEC0::3:0/120 [1/0], tag 1
     via Tunnel0, directly connected
S   FEC0::10:3/128 [1/0], tag 1
     via Tunnel0, directly connected
```

The FlexVPN GRE/IPv6 tunnel carries both IPv4 and IPv6 traffic to the FlexVPN hub LAN subnets learned through FlexVPN routing.

Click here to view code image

```
FlexSpoke2#clear crypto sa counters
FlexSpoke2#show crypto session detail | begin IPSEC
  IPSEC FLOW: permit 47 host 2001::1:1 host 2001::1:2
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 0 drop 0 life (KB/Sec) 4346787/3407
        Outbound: #pkts enc'ed 0 drop 0 life (KB/Sec) 4346787/3407

FlexSpoke2#ping ipv6 FEC0::3:1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to FEC0::3:1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexSpoke2#show crypto session detail | begin IPSEC
  IPSEC FLOW: permit 47 host 2001::1:1 host 2001::1:2
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 5 drop 0 life (KB/Sec) 4346787/3387
        Outbound: #pkts enc'ed 5 drop 0 life (KB/Sec) 4346786/3387

FlexSpoke2#ping 192.168.2.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.2.1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms
```

```
FlexSpoke2#show crypto session detail | begin IPSEC
  IPSEC FLOW: permit 47 host 2001::1:1 host 2001::1:2
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 10 drop 0 life (KB/Sec) 4346785/3361
        Outbound: #pkts enc'ed 10 drop 0 life (KB/Sec) 4346785/3361
```

Verification on the FlexVPN Hub at HQ

The FlexVPN hub clones the virtual-access interface for FlexVPN spoke-1 and spoke-2 from the same virtual-template but automatically detects the encapsulation mode and transport IP type from the incoming session and sets the **tunnel mode** accordingly.

FlexVPN session and the virtual-access interface for FlexVPN spoke-1 that uses GRE/IPv4.

Click here to view code image

```
FlexHub#show crypto session
Crypto session current status
Interface: Virtual-Access2
Profile: default
Session status: UP-ACTIVE
Peer: 172.16.1.1 port 500
  Session ID: 5
  IKEv2 SA: local 172.16.1.2/500 remote 172.16.1.1/500 Active
  IPSEC FLOW: permit 47 host 172.16.1.2 host 172.16.1.1
        Active SAs: 2, origin: crypto-map
FlexHub#show derived-config interface virtual-access 2
Derived configuration : 220 bytes

interface Virtual-Access2
 ip unnumbered Loopback1
 ipv6 unnumbered Loopback1
 tunnel source 172.16.1.2
 tunnel destination 172.16.1.1
 tunnel protection ipsec profile default
 no tunnel protection ipsec initiate
end

FlexHub#show interfaces virtual-access 2 | include protocol/transport
  Tunnel protocol/transport GRE/IP
```

The FlexVPN session and the virtual-access interface for FlexVPN spoke 2 that uses GRE/IPv6.

Click here to view code image

```
FlexHub#show crypto session
Crypto session current status
Interface: Virtual-Access1
Profile: default
Session status: UP-ACTIVE
Peer: 2001::1:1 port 500
  Session ID: 6
  IKEv2 SA: local 2001::1:2/500
          remote 2001::1:1/500 Active
  IPSEC FLOW: permit 47 host 2001::1:2 host 2001::1:1
        Active SAs: 2, origin: crypto-map

FlexHub#show derived-config interface virtual-access 1
Derived configuration : 240 bytes
```

```
interface Virtual-Access1
 ip unnumbered Loopback1
 ipv6 unnumbered Loopback1
 tunnel source 2001::1:2
 tunnel mode gre ipv6
 tunnel destination 2001::1:1
 tunnel protection ipsec profile default
 no tunnel protection ipsec initiate
end

FlexHub#show interfaces virtual-access 1 | include protocol/transport
  Tunnel protocol/transport GRE/IPv6
```

The FlexVPN hub learns the IPv4 and IPv6 LAN subnets from FlexVPN spokes 1 and 2 and adds static routes to those subnets through the corresponding virtual-access interface.

Click here to view code image

```
FlexHub#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA
Tunnel-id Local                 Remote                fvrf/ivrf             Status
1         172.16.1.2/500        172.16.1.1/500        none/none             READY
      Remote subnets:
      10.0.0.1 255.255.255.255
      192.168.0.0 255.255.255.0
      IPv6 Remote subnets:
      FEC0::10:1/128
      FEC0::1:0/120

 IPv6 Crypto IKEv2  SA
Tunnel-id    fvrf/ivrf              Status
2          none/none              READY
Local  2001::1:2/500
Remote  2001::1:1/500
            Remote subnets:
      10.0.0.2 255.255.255.255
      192.168.1.0 255.255.255.0
      IPv6 Remote subnets:
      FEC0::10:2/128
      FEC0::2:1/120

FlexHub#show ip route static
      10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
S        10.0.0.1/32 is directly connected, Virtual-Access2
S        10.0.0.2/32 is directly connected, Virtual-Access1
S     192.168.0.0/24 is directly connected, Virtual-Access2
S     192.168.1.0/24 is directly connected, Virtual-Access1

FlexHub#show ipv6 route static
S   FEC0::1:0/120 [1/0], tag 1
     via Virtual-Access2, directly connected
S   FEC0::2:0/120 [1/0], tag 1
     via Virtual-Access1, directly connected
S   FEC0::10:1/128 [1/0], tag 1
     via Virtual-Access2, directly connected
S   FEC0::10:2/128 [1/0], tag 1
     via Virtual-Access1, directly connected
```

The FlexVPN hub can send and receive both IPv4 and IPv6 traffic to the FlexVPN spokes over the GRE/IPv4 and GRE/IPv6 tunnels.

Traffic to FlexVPN spoke 1:

Click here to view code image

```
FlexHub#ping 192.168.0.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.0.1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexHub#ping ipv6 FEC0::1:1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to FEC0::1:1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexHub#show crypto session interface virtual-access 2 detail | begin IPSEC
  IPSEC FLOW: permit 47 host 172.16.1.2 host 172.16.1.1
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 10 drop 0 life (KB/Sec) 4200943/2000
        Outbound: #pkts enc'ed 10 drop 0 life (KB/Sec) 4200944/2000
```

Traffic to FlexVPN spoke 2:

Click here to view code image

```
FlexHub#ping 192.168.1.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexHub#ping ipv6 FEC0::2:1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to FEC0::2:1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexHub#show crypto session interface virtual-access 1 detail | begin IPSEC
  IPSEC FLOW: permit 47 host 2001::1:2 host 2001::1:1
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 10 drop 0 life (KB/Sec) 4339994/2178
        Outbound: #pkts enc'ed 10 drop 0 life (KB/Sec) 4339996/2178
```

FlexVPN Client NAT to the Server-Assigned IP Address

This scenario demonstrates the FlexVPN client applying NAT source address translation to the LAN traffic that is sent through the FlexVPN tunnel. The source address of the LAN packets is translated to the address assigned by the FlexVPN server. This scenario is equivalent of the EzVPN client mode. The following example illustrates the NAT configuration on the FlexVPN client.

Configuration on the FlexVPN Client

The AAA authorization is configured to use local AAA for FlexVPN group authorization.

Click here to view code image

```
aaa new-model

aaa authorization network group_author_list local

crypto ikev2 name-mangler group_author_name_mangler
 fqdn domain
```

The IKEv2 authorization policy, which is used for local group authorization in this example, is configured to accept the routes pushed by the peer and to push the VPN tunnel IP address to the peer.

Click here to view code image

```
crypto ikev2 authorization policy default
 route set interface

crypto ikev2 profile default
 match identity remote fqdn hq.example.com
 identity local fqdn branch1.example.com
 authentication local pre-share key cisco
 authentication remote pre-share key cisco
 aaa authorization group psk list group_author_list default

crypto ikev2 client flexvpn flex_client_profile
  peer 1 172.16.1.2
  connect manual
  client connect Tunnel0
```

The loopback interface configured with **ip nat inside** simulates the LAN subnet for which NAT translation is applied. In production networks, the interface will be a routable physical interface.

Click here to view code image

```
interface Loopback0
 description simulates LAN subnet
 ip address 192.168.0.1 255.255.255.0
 ip nat inside
 ip virtual-reassembly in
```

The **ip nat outside**, combined with the NAT rule defined in the following example, applies network access translation to all the LAN traffic forwarded to tunnel interface. Note that the translation is done before encryption. The source address of all the packets is translated to the IP address of the tunnel interface which is obtained from the FlexVPN server through IKEv2 configuration exchange due to the **ip address negotiated** command.

Click here to view code image

```
interface Tunnel0
 ip address negotiated
 ip nat outside
 ip virtual-reassembly in
 tunnel source Ethernet0/0
 tunnel destination dynamic
 tunnel protection ipsec profile default

interface Ethernet0/0
 description WAN interface
 ip address 172.16.1.1 255.255.255.0
```

Define the NAT rule to apply source address translation to traffic forwarded to the tunnel interface.

The source address of all the packets is translated to the IP address of the tunnel interface.

Click here to view code image

```
ip nat inside source route-map flex-route-map interface Tunnel0 overload

route-map flex-route-map permit 1
 match interface Tunnel0
```

Verification on the FlexVPN Client

The following commands display the IP address assigned by the FlexVPN server and the remote LAN subnets pushed by the server. The client adds routes to the remote subnets.

Click here to view code image

```
FlexClient#show crypto ikev2 sa detail
 IPv4 Crypto IKEv2  SA
Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         172.16.1.1/500        172.16.1.2/500        none/none            READY
      Pushed IP address: 10.0.0.1
      DNS Primary: 192.168.10.10
      Remote subnets:
      10.0.0.10 255.255.255.255
      192.168.1.0 255.255.255.0

FlexClient#show ip route static
      4.0.0.0/24 is subnetted, 1 subnets
S        4.4.4.0 is directly connected, Tunnel0
      10.0.0.0/8 is variably subnetted, 3 subnets, 2 masks
S        10.0.0.10/32 is directly connected, Tunnel0
S     192.168.1.0/24 is directly connected, Tunnel0
```

The following **debug ip packet** logs capture the traffic from the local to the remote LAN that is forwarded to the tunnel interface where the packet source address is translated to the server-assigned IP address by NAT and then the packet is encrypted and sent out.

Click here to view code image

```
FlexClient#ping 192.168.1.1 source 192.168.0.1 repeat 1
Type escape sequence to abort.
Sending 1, 100-byte ICMP Echos to 192.168.1.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.0.1

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexClient#
IP: s=192.168.0.1 (local), d=192.168.1.1, len 100, local feature, NAT(2), rtype 0,
forus FALSE, sendself FALSE, mtu 0, fwdchk FALSE
IP: s=192.168.0.1 (local), d=192.168.1.1 (Tunnel0), len 100, sending
```

NAT translation is applied.

Click here to view code image

```
IP: s=10.0.0.1 (local), d=192.168.1.1 (Tunnel0), len 100, output feature, Post-
routing NAT Outside(26), rtype 1, forus FALSE, sendself FALSE, mtu 0, fwdchk FALSE
```

Packet undergoes GRE encapsulation and encryption.

Click here to view code image

```
IP: s=10.0.0.1 (local), d=192.168.1.1 (Tunnel0), len 100, post-encap feature,
IPSEC Post-encap output classification(16), rtype 1, forus FALSE, sendself FALSE,
mtu 0, fwdchk FALSE
IP: s=10.0.0.1 (local), d=192.168.1.1 (Tunnel0), len 100, sending full packet
IP: s=172.16.1.1 (local), d=172.16.1.2 (Ethernet0/0), g=172.16.1.2, len 168,
forward
IP: s=172.16.1.1 (local), d=172.16.1.2 (Ethernet0/0), len 168, sending full packet
```

The **show ip nat translations** and **show ip nat statistics** command outputs display the NAT translation entries.

Click here to view code image

```
FlexClient#show ip nat translations
Pro Inside global      Inside local       Outside local      Outside global
icmp 10.0.0.1:15       192.168.0.1:15     192.168.1.1:15     192.168.1.1:15

FlexClient#show ip nat statistics
Total active translations: 1 (0 static, 1 dynamic; 1 extended)
Peak translations: 4, occurred 01:13:02 ago
Outside interfaces:
  Tunnel0
Inside interfaces:
  Loopback0
Hits: 65  Misses: 0
CEF Translated packets: 30, CEF Punted packets: 0
Expired translations: 13
Dynamic mappings:
-- Inside Source
[Id: 1] route-map flex-route-map interface Tunnel0 refcount 1
```

FlexVPN WAN Resiliency, Using Dynamic Tunnel Source

This scenario demonstrates FlexVPN WAN resiliency using the FlexVPN client dynamic tunnel source feature. The example uses a branch site connected to the headquarters over MPLS and the Internet as primary and secondary WAN links as illustrated in Figure 12-3.

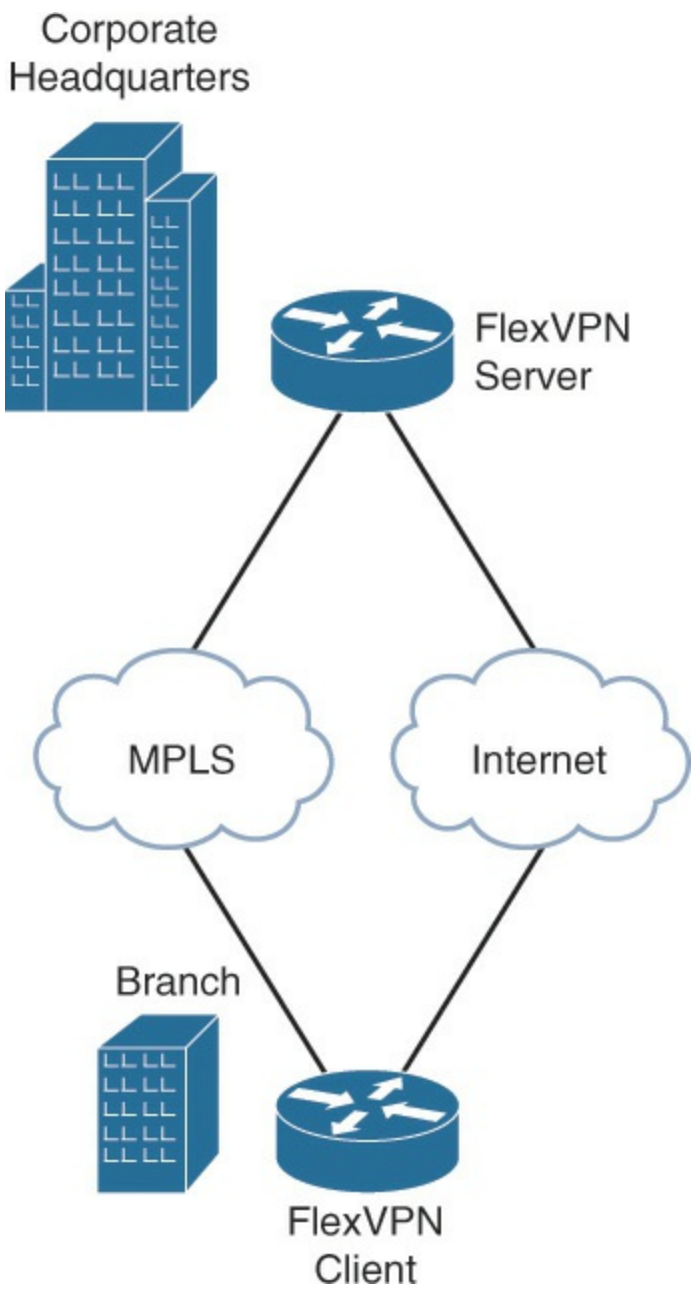


Figure 12-3 *Dual-Homed FlexVPN Client and Server*

The dual-homed branch router is configured as the FlexVPN client and the dual-homed edge router at the head end site is configured as a FlexVPN server. The FlexVPN client would connect to the FlexVPN server over MPLS as long as it is available, fall back to the Internet when the MPLS link goes down, and revert back to MPLS when it comes back up. The transport networks don't strictly have to be MPLS or Internet, but they could be 3G, 4G, satellite, or any bearer that runs IP. This feature is especially useful with expensive, usage-based backup links such as 3G, 4G, or satellite where the backup link is used only during the time the primary link is unavailable.

The following example illustrates the FlexVPN WAN resiliency configuration and verification on the FlexVPN client.

FlexVPN Client Configuration on the Dual-Homed Branch Router

The Serial 0/0 and Serial 1/0 interfaces in this example simulate MPLS and Internet WAN links. The track objects track the line protocol status of the two serial interfaces.

Click here to view code image

```
track 1 interface Serial0/0 line-protocol
```

```
track 2 interface Serial1/0 line-protocol
```

The FlexVPN client profile is configured to track the line protocol status of the Serial 0/0 and Serial 1/0 interfaces and use them as the preferred and backup tunnel source interface. The peer address is the address of the loopback interface on the FlexVPN server that is reachable through both Serial 0/0 and Serial 1/0 interfaces. This example is intended to be very simple in nature. The ability to track objects allows for very powerful configurations to be used. Rather than simply monitoring the line protocol, a track could be created to monitor for the presence of a prefix in the routing table, or an SLA which can check connectivity to a next-hop IP address.

The profile is configured with the default **connect auto** mode to be able to initiate the FlexVPN connection without any user intervention when there is a need to switch between the preferred and backup tunnel source interfaces, based on their availability.

Click here to view code image

```
crypto ikev2 client flexvpn flex-client
  peer 1 172.16.3.1
  source 1 Serial0/0 track 1
  source 2 Serial1/0 track 2
  client connect Tunnel0
```

The tunnel interface is associated with the FlexVPN client profile, using the **client connect Tunnel0** command in the client profile. With the **tunnel source dynamic** command, the tunnel source is dynamically set by the client profile, based on the availability of the source interfaces specified in the profile.

Click here to view code image

```
interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 ipv6 address FEC0::10:1/120
 tunnel source dynamic
 tunnel destination dynamic
 tunnel protection ipsec profile default
```

The Serial 0/0 and Serial 1/0 interfaces in this example simulate MPLS and Internet WAN links.

Click here to view code image

```
interface Serial0/0
 description MPLS
 ip address 172.16.1.1 255.255.255.0
 keepalive 2
 serial restart-delay 0

interface Serial1/0
 description Internet
 ip address 172.16.2.1 255.255.255.0
 keepalive 2
 serial restart-delay 0
```

The FlexVPN server loopback address is configured to be reachable through both Serial 0/0 and Serial 1/0 interfaces.

Click here to view code image

```
ip route 172.16.3.1 255.255.255.255 Serial0/0
ip route 172.16.3.1 255.255.255.255 Serial1/0
```

Verification on the FlexVPN Client

FlexVPN client is initially connected to the FlexVPN server with Serial 0/0 as tunnel source interface.

Click here to view code image

```
FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:ACTIVE
  Peer : 172.16.3.1
  Source : Serial0/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel0
```

When the Serial 0/0 interface goes down, the associated track object and the FlexVPN connection go down as captured in the following syslogs.

Click here to view code image

```
%TRACK-6-STATE: 1 interface Se0/0 line-protocol Up -> Down
* %FLEXVPN-6-FLEXVPN_CONNECTION_DOWN: FlexVPN(flex-client) Client_public_addr =
172.16.1.1 Server_public_addr = 172.16.3.1
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to down
%CRYPTO-6-ISAKMP_ON_OFF: ISAKMP is OFF
%CRYPTO-6-ISAKMP_ON_OFF: ISAKMP is ON
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0, changed state to down
```

The FlexVPN profile uses the next configured and available interface (Serial 1/0) as tunnel source and re-establishes the FlexVPN connection.

Click here to view code image

```
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up
%FLEXVPN-6-FLEXVPN_CONNECTION_UP: FlexVPN(flex-client) Client_public_addr =
172.16.2.1 Server_public_addr = 172.16.3.1

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:ACTIVE
  Peer : 172.16.3.1
  Source : Serial1/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel0
```

When the Serial 0/0 interface comes back up, the FlexVPN connection switches back to Serial 0/0 as the tunnel source interface.

Click here to view code image

```
TRACK-6-STATE: 1 interface Se0/0 line-protocol Down -> Up
%FLEXVPN-6-FLEXVPN_CONNECTION_DOWN: FlexVPN(flex-client) Client_public_addr =
172.16.2.1 Server_public_addr = 172.16.3.1
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to down
```

```
%CRYPTO-6-ISAKMP_ON_OFF: ISAKMP is OFF
%CRYPTO-6-ISAKMP_ON_OFF: ISAKMP is ON

%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0, changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up
%FLEXVPN-6-FLEXVPN_CONNECTION_UP: FlexVPN(flex-client) Client_public_addr =
172.16.1.1 Server_public_addr = 172.16.3.1

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:ACTIVE
  Peer : 172.16.3.1
  Source : Serial0/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel0
```

FlexVPN Hub Resiliency, Using Backup Peers

This scenario demonstrates FlexVPN hub resiliency, using the FlexVPN client backup peers feature. The example uses a branch site connected to the headquarters that has dual WAN edge routers for redundancy as illustrated in Figure 12-4.

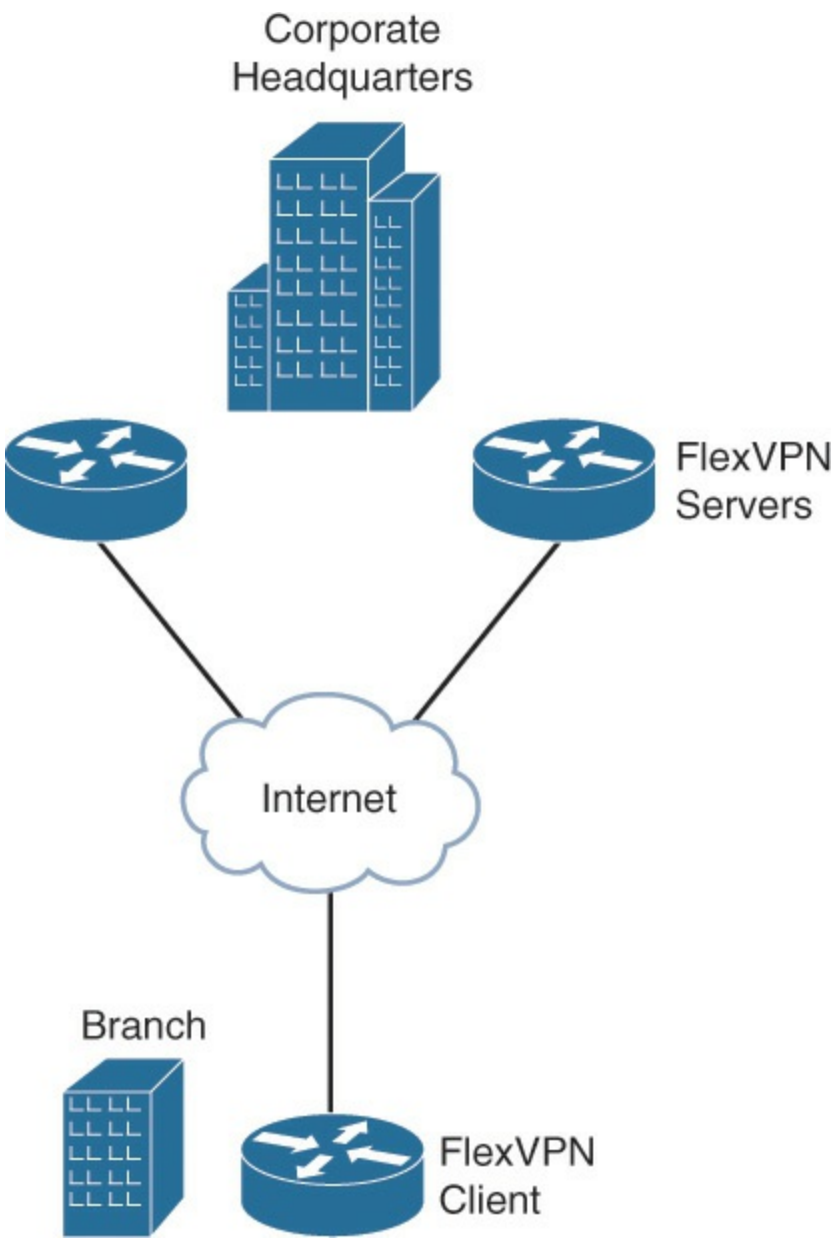


Figure 12-4 *Dual FlexVPN Hub*

The branch router is configured as FlexVPN client, and each of the edge routers at the headquarters is configured as a FlexVPN hub. The FlexVPN client is configured with primary and secondary hubs as peers. The client would connect to the primary hub, based on a track object that tracks its reachability and would fall back to the backup FlexVPN hub only when the preferred hub is not available, reverting back to the primary once it is available. The following example illustrates the configuration on the FlexVPN client for FlexVPN hub resiliency, using backup peers.

FlexVPN Client Configuration on the Branch Router

A track object is configured to track the reachability of the primary hub.

Click here to view code image

```
track 1 ip sla 1 reachability

ip sla 1
 icmp-echo 172.16.2.1
 frequency 5
ip sla schedule 1 start-time now
```

The FlexVPN client profile is configured with primary and secondary peers. The client would

connect to the primary peer when the associated track object that tracks its reachability is up and would fall back to secondary only when the primary is not reachable. The **peer reactivate** command ensures that the connection reverts to the primary when it is available again. The profile is configured with the default **connect auto** mode to be able to initiate the FlexVPN connection without any user intervention when there is a need to switch between the primary and secondary peers based on the availability of the primary.

Click here to view code image

```
crypto ikev2 client flexvpn flex-client
  peer 1 172.16.2.1 track 1
  peer 2 172.16.3.1
  peer reactivate
  client connect Tunnel0
```

The tunnel interface is associated with the FlexVPN client profile, using the **client connect Tunnel0** in the client profile. With the **tunnel destination dynamic** command, the tunnel destination is dynamically set by the client profile, based on the availability of the primary and secondary peers.

Click here to view code image

```
interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination dynamic
 tunnel protection ipsec profile default
```

The Ethernet 0/0 connects to the Internet that provides connectivity to the primary and secondary peers.

Click here to view code image

```
interface Ethernet0/0
 description Internet
 ip address 172.16.1.1 255.255.255.0
```

Verification on the FlexVPN Client

The FlexVPN client is initially connected to the primary peer.

Click here to view code image

```
FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:ACTIVE
  Peer : 172.16.2.1
  Source : Ethernet0/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel0
```

When the primary peer becomes unreachable, the associated track object and the FlexVPN connection go down as captured in the following syslogs.

Click here to view code image

```
%TRACK-6-STATE: 1 ip sla 1 reachability Up -> Down
%FLEXVPN-6-FLEXVPN_CONNECTION_DOWN: FlexVPN(flex-client) Client_public_addr =
```

```
172.16.1.1 Server_public_addr = 172.16.2.1
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to down
```

The client establishes the connection to the secondary peer.

Click here to view code image

```
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up
%FLEXVPN-6-FLEXVPN_CONNECTION_UP: FlexVPN(flex-client) Client_public_addr =
172.16.1.1 Server_public_addr = 172.16.3.1

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:ACTIVE
  Peer : 172.16.3.1
  Source : Ethernet0/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel0
```

When the primary peer comes back up, the FlexVPN connection switches back to the primary.

Click here to view code image

```
%TRACK-6-STATE: 1 ip sla 1 reachability Down -> Up
%FLEXVPN-6-FLEXVPN_CONNECTION_DOWN: FlexVPN(flex-client) Client_public_addr =
172.16.1.1 Server_public_addr = 172.16.3.1
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to down

%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up
%FLEXVPN-6-FLEXVPN_CONNECTION_UP: FlexVPN(flex-client) Client_public_addr =
172.16.1.1 Server_public_addr = 172.16.2.1

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:ACTIVE
  Peer : 172.16.2.1
  Source : Ethernet0/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel0
```

FlexVPN Backup Tunnel, Using Track-Based Tunnel Activation

This scenario demonstrates FlexVPN tunnel as a backup WAN connection, using the FlexVPN client track-based tunnel activation feature. The example uses a branch site connected to the headquarters over MPLS and Internet with MPLS as the primary WAN link and the FlexVPN tunnel over the Internet as the secondary WAN link. The FlexVPN client is configured to track the MPLS link, and when it goes down, the FlexVPN tunnel over the Internet would be activated to act as a backup WAN link. The FlexVPN tunnel is terminated once the MPLS link is available again. The following example illustrates the configuration of the FlexVPN backup tunnel, using track-based tunnel activation.

The Serial 1/0 interface in this example simulates the MPLS WAN link. The track object tracks the line protocol status of the Serial 1/0 interface.

Click here to view code image

```
track 1 interface Serial1/0 line-protocol
```

The FlexVPN client profile is configured to initiate the FlexVPN tunnel when the track object 1 goes down. As the track object 1 monitors the availability of the MPLS link, the FlexVPN tunnel is triggered only when the MPLS interface is down.

Click here to view code image

```
crypto ikev2 client flexvpn flex-client
  peer 1 172.16.1.2
  connect track 1 down
  client connect Tunnel0

interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination dynamic
 tunnel protection ipsec profile default
```

The Ethernet 0/0 and Serial 1/0 interfaces in this example simulate Internet and MPLS WAN links.

Click here to view code image

```
interface Ethernet0/0
 description Internet
 ip address 172.16.1.1 255.255.255.0

interface Serial1/0
 description MPLS
 ip address 172.16.2.1 255.255.255.0
 keepalive 2
 serial restart-delay 0
```

Verification on the FlexVPN Client

The FlexVPN tunnel is initially inactive as the MPLS link is available.

Click here to view code image

```
FlexClient#show interfaces Serial 1/0 | include line protocol
Serial1/0 is up, line protocol is up

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:CONNECT_REQUIRED
  Backup group: Default
  Tunnel interface : Tunnel0
```

When the MPLS link (Serial 1/0 interface) goes down, the associated track object goes down, and the FlexVPN tunnel over the Internet is initiated as captured in the following syslog messages.

Click here to view code image

```
%TRACK-6-STATE: 1 interface Se1/0 line-protocol Up -> Down
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up
%FLEXVPN-6-FLEXVPN_CONNECTION_UP: FlexVPN(flex-client) Client_public_addr =
172.16.1.1 Server_public_addr = 172.16.1.2

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:ACTIVE
```

```
  Peer : 172.16.1.2
  Source : Ethernet0/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel0
```

When the MPLS link (Serial 1/0 interface) comes back up, the FlexVPN tunnel is terminated.

Click here to view code image

```
%TRACK-6-STATE: 1 interface Se1/0 line-protocol Down -> Up
%FLEXVPN-6-FLEXVPN_CONNECTION_DOWN:FlexVPN(flex-client) Client_public_addr =
172.16.1.1 Server_public_addr = 172.16.1.2

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:CONNECT_REQUIRED
  Backup group: Default
  Tunnel interface : Tunnel0
```

Summary

This chapter illustrated some of the FlexVPN deployment scenarios with sample configuration and verification steps. The scenario on AAA-based pre-shared keys illustrated how the RADIUS server can be leveraged for scalable management of pre-shared keys, especially at the headend that needs to deal with large number of peers.

The scenario with FlexVPN authorization illustrated how the user and group authorization can be combined together to enforce specific and common authorization policy across the sites. The scenario on dual stack and tunnel mode auto illustrated how these features simplify the deployments while accomplishing the required functionality of IPv4 and IPv6 overlay over IPv4 or IPv6 transport. The scenario also demonstrated how FlexVPN routing simplifies overlay routing.

The chapter concluded with FlexVPN NAT and the various FlexVPN resiliency scenarios.

Part VI: IPsec VPN Maintenance

Chapter 13. Monitoring IPsec VPNs

Introduction to Monitoring

Configuring IPsec VPNs can be seen by some as a cumbersome process. In many instances, once an IPsec VPN is configured it is monitored by the IP connectivity that runs within the overlay network, protected by the VPN. This isn't the only method to determine if the VPN has been successfully created and is working correctly. This chapter describes the method to monitor each process that is performed when establishing an IPsec Security Association using IKEv2. This chapter is structured so that the method used to monitor each function that is used to establish an IPsec session is described in an ordered fashion. Each function builds on the previous one; for example, IP connectivity must exist between the source and destination of each peer before the IKE exchange can occur.

When the IPsec architecture is monitored, the overall process can be separated into control-plane and data-plane processes. The control-plane processes are used to establish and build the IPsec Security Association. Data-plane monitoring is used to gain insight into the traffic and services that passes over the IPsec Security Association.

The telemetry that is generated by Cisco IOS can be viewed locally in the case of syslog or local NetFlow statistics. For the majority of enterprise architectures, a Security Information Event Management (SIEM) system is used to correlate logging and monitoring information to determine security related events. This chapter will describe how to generate this telemetry but not how to operate a SIEM.

The following protocols are used in the monitoring of IPsec VPNs.

Authentication, Authorization, and Accounting (AAA)

AAA is made up of three services:

- **Authentication.** Authentication identifies users; its mechanisms include login and password dialog, challenge and response, messaging support, and, depending on the security protocol that you select, encryption.

  In the context of IKEv2, authentication is the service that verifies the identity of the user or device within the IKE_AUTH exchange. Authentication is based on the IKE ID and PSK or certificate, or EAP identity and authentication method (password/certificate). AAA on Cisco IOS devices allows you to perform local authentication (using the local lookup database) or remote authentication (using one or more RADIUS servers).

- **Authorization.** Authorization is the service that describes the attributes as to what the user is authorized to perform. Authorization is provided by attributes that are obtained from a AAA server or locally. Attribute-Value (AV) pairs are used to contain the specific rights of the appropriate user.

- **Accounting.** Accounting provides the service for collecting information, logging the information locally, and sending the information to the AAA server for auditing and reporting purposes.

  The accounting service tracks and maintains a log of every session used to access the VPN headend.

NetFlow

NetFlow comes in a number of varieties; Flexible NetFlow has many benefits over the traditional Cisco NetFlow functionality that has been available for years in Cisco hardware and software.

Flexible NetFlow is integral part of Cisco IOS software that collects and measures data allowing all routers or switches in the network to become a source of telemetry and a monitoring device. Flexible NetFlow provides extremely granular and accurate traffic measurements, resulting in rich telemetry about the type of traffic being monitored.

NetFlow allows users to monitor a wide range of packet information including packet headers and details of flows. These values are user configurable, allowing for customized traffic identification and the ability to focus and monitor specific network behavior.

Data is sent from flow exporters to flow collectors. The flow collectors can then be queried by analysis applications to give insight into traffic profiling.

NetFlow is generally used to gain insight into the traffic traversing within the IPsec Security Association. Hence this is a data-plane monitoring. This can be achieved by applying the NetFlow policy to the tunnel interface. In this instance details of all flows that are sent within the IPsec Security Association will be captured. The flow data will contain details of the specified information such as client to server IP addresses, protocols, and ports.

When the NetFlow policy is applied to the tunnel source interface, IKE traffic and encrypted traffic are monitored. The concept of viewing details of the IKE traffic is classified as control-plane monitoring. When NetFlow is applied to the source of the tunnel interface, telemetry about the encapsulated and protected traffic can be gathered. In this case, the flow data will only contain details of the IKE and IPsec flows and hence there will be far fewer flows than when a NetFlow policy is applied to the tunnel interface.

Simple Network Management Protocol

Simple Network Management Protocol (SNMP) is an application layer protocol that supports communication between SNMP managers and agents. SNMP provides a standardized framework and a common language used for the monitoring and management of devices in a network.

The SNMP framework is composed of the following three components:

- **SNMP Manager.** An SNMP manager is the system used to control and monitor the activities of network hosts using SNMP. The most common managing system is called a Network Management System (NMS). The term NMS can be applied to either a dedicated device used for network management, or the applications used on such a device.
- **SNMP Agent.** The SNMP agent is the software component within the managed device that maintains the data for the device and reports the data, as needed, to managing systems. The agent and MIB reside on the routing device (router, access server, or switch). To enable the SNMP agent on a Cisco routing device, you must define the relationship between the manager and the agent.
- **Management Information Base (MIB).** The MIB is a virtual information storage area for network management information, which consists of collections of managed objects. Within the MIB, there are collections of related objects, defined in MIB modules. MIB modules are written in the SNMP MIB module language, as defined in RFC2578, RFC2579, and RFC2580.

The SNMP agent contains MIB variables whose values the SNMP manager can request or change through get or set operations. A manager can get a value from an agent or store a value into that agent. The agent gathers data from the MIB, the repository for information about device parameters and network data. The agent can also respond to manager requests to get or set data. The agent can send unsolicited data using a trap or inform operation.

Three different versions of SNMP can be configured:

- **SNMPv1.** SNMPv1 utilizes a community-based security mechanism and is widely considered to be insecure.
- **SNMPv2c.** SNMPv2c utilizes a community-based security mechanism and is nearly identical to SNMPv1 in many instances. The counters are set to 64 bits, compared to 32 bits within v1.
- **SNMPv3.** SNMPv3 integrated a number of security features (known as security models) to overcome the weaknesses present in v1 and v2c. SNMPv3 provides authentication, encryption, and message integrity.

SNMP uses the UDP protocol over port 161 for listening to get or set requests and sends trap or inform requests to port 162.

SNMP messages are relatively computationally expensive to produce because of ASN.1 encoding. SNMP should not be used when normal conditions may generate a large amount of messages because it could incur an excessive load . SNMP is better suited for alerts when abnormal conditions occur.

VRF-Aware SNMP

Cisco IOS allows for a number of MIBs to be VRF aware. VRF-aware SNMP allows for a MIB interface linked to a MIB to reside in one VRF; however, the generated trap can be sent to a SNMP server that resides in a separate VRF.

This architecture allows a FVRF to be used to generate IKE and IPsec related information. This can be sent to a SNMP server that resides in the management VRF (or any VRF that is separate to the FVRF).

The following example illustrates how to configure VRF-aware SNMP for a VRF-aware IPsec deployment.

1. An SNMP context is created.

Click here to view code image

```
snmp-server context SNMP-WAN
```

2. A VRF is created with the associated SNMP context.

Click here to view code image

```
vrf definition wan
    address-family ipv4
     snmp context  SNMP-WAN
    exit-address-family
```

3. An SNMP group is created and associated with the SNMP context.

Click here to view code image

```
snmp-server group wan-grp   v3 auth context SNMP-WAN
```

4. A SNMP user is assigned to a group.

Click here to view code image

```
snmp-server user wan-user wan-grp v3 auth sha cisco123
```

5. The SNMP server is defined, using the user account created previously.

Click here to view code image

```
snmp-server host 10.10.10.1 vrf mgmt version 3 auth  wan-user
```

Using this configuration, traps can be generated for events that occur within the FVRF, or the IVRF, but they will be sent to an SNMP server.

Syslog

RFC 3164, "The BSD syslog Protocol" describes the syslog protocol, which provides a transport mechanism to allow a machine to send event notification messages across IP networks to event message collectors, also known as syslog servers.

Each message will contain a severity value; Table 13-1 illustrates these levels and the meaning of each.

Numerical Code	Severity
0	Emergency: system is unusable
1	Alert: action must be taken immediately
2	Critical: critical conditions
3	Error: error conditions
4	Warning: warning conditions
5	Notice: normal but significant conditions
6	Informational: informational messages
7	Debug: debug-level messages

Table 13-1 *Syslog Severity Levels*

The syslog server will run a syslog daemon, which commonly listens on UDP port 514 or TCP port 601 when a reliable syslog configuration is used. The syslog sender will send a message of 1KB or less. Because it is a connectionless protocol, UDP does not provide acknowledgments to the sender or receiver. Additionally, at the application layer, syslog servers do not send acknowledgments back to the sender for receipt of syslog messages. Consequently, the sending device generates syslog messages without knowing whether the syslog server has received the messages. In fact, the sending devices will send messages even if the syslog server does not exist; these messages get lost in the network.

A syslog message seen on the wire has three distinct parts: the PRI (priority), HEADER, and MSG (message).

The total length of the packet cannot exceed 1024 bytes. There is no minimum length.

A syslog service can be verbose in nature, so it must be implemented precisely. Adequate thresholds

and filters must be defined to generate actionable alerts based on the syslog messages. Critical syslog messages should be prioritised easily, resulting in benign messages not alerting staff to non-critical events. The generation of a large number of syslog messages can cause excessive control-plane activity.

Monitoring Methodology

When a methodology is applied to monitoring IPsec VPNs, a structured approach can be taken to ensure that untoward activities are identified and subsequently rectified in a timely manner.

Understanding the conditions that occur when an IPsec VPN is correctly functioning and the conditions that occur when an IPsec is not functioning correctly is the key to success when deploying a monitoring solution. Knowing that an IPsec Security Association has not been created successfully is as critical as knowing when an IPsec Security Association is functioning correctly.

The ideal monitoring solution would give enough information to monitor what is considered normal activity; however, when abnormal activity occurs more verbose telemetry would be generated.

Table 13-2 illustrates the functions that are used to establish an IPsec Security Association, the corresponding protocol, and the method that can be used when monitoring connectivity.

Function	Protocol	Monitoring Method
IP connectivity	Transport routing protocol	SNMP, syslog
VPN tunnel establishment	IKEv2	SNMP, syslog
Authentication	PSK	AAA, syslog
Authentication	PKI	syslog
Authentication	EAP	AAA, syslog
Authorization	AAA, RADIUS	AAA, syslog
Data encryption	IPsec	SNMP, AAA accounting
Overlay routing	Overlay routing protocol IKEv2 routing	SNMP, syslog
Data usage	IPsec	NetFlow

Table 13-2 *Monitoring Methodology*

When a monitoring architecture is deployed, attention should be paid to components that can cause the critical functions to fail. For example, if an IPsec headend is being used for remote access connections that uses EAP with a backend RADIUS server, should connectivity fail to the RADIUS server, all new user sessions will fail to authenticate. This is in contrast to the same architecture, however should the attributes of a single user fail to be retrieved only the user session will fail. This will have considerably less impact to the overall system than a failure condition that affects all users such as loss of connectivity to the RADIUS server.

Whenever possible, automated monitoring should occur. By this we mean that an automated mechanism should issue an alert if critical components fail and the alert should be sent with meaningful data. If possible, verbose logging should occur only when a critical component fails, for

this minimizes the impact that logging can have on the CPU, available bandwidth, and storage. It is also prudent to alert only when critical events occur that will cause major impact. It would be unwise to page an on-call engineer at 02:00 a.m. because a single IKE session failed to authenticate for a noncritical system. However, should a major component fail that will affect all users of a service, such as CRL retrieval if revocation is enabled or loss of connectivity to a RADIUS server when EAP is in use, then a critical event should be generated.

Any data that is generated by monitoring can be useful; in many instances it is the only evidence after an event has occurred.

Please be mindful of enabling extremely verbose logging, in many instances this can generate a large amount of CPU cycles. There have been numerous cases where a large number of syslog servers were configured in conjunction with verbose logging. The result was constant high CPU as many syslog messages were sent to the numerous syslog servers.

For any messages that are generated locally and include a timestamp, Network Time Protocol (NTP) should be used to ensure accurate time reporting. Time should be synchronized from a trusted source with authentication enabled.

The following example illustrates enabling timestamps for logging and debugging messages, in addition to setting the time zone. This will ensure that all messages can be accurately identified in a timely fashion.

Click here to view code image

```
service timestamps debug datetime localtime show-timezone
service timestamps log datetime localtime show-timezone
clock timezone GMT 0
```

IP Connectivity

Before IP connectivity can occur, the interface that is sending and receiving data must be in the up state. This means that correct cabling must be present and that there is an active line protocol. Issues that could cause an interface to not be in the up state include misconfigured cabling, broken fibers, and hardware issues with network interface cards (NIC).

Knowing if the interface that is acting as the source of a tunnel is flapping (going up and down), or even changed state can be critical in helping to determine connectivity issues. The following example illustrates the logging messages that are generated when an interface changes state.

Click here to view code image

```
%LINEPROTO-5-UPDOWN: Line protocol on Interface Ethernet0/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface Ethernet0/0, changed state to down
```

Interfaces that are critical for the functionality of the system should be monitored for any indication of flapping. For interfaces that do not require monitoring, the **no logging event link-status** command can be configured on the interface to turn off the generation of messages when the state changes.

SNMP can be used to monitor when state of physical interfaces change. The following example illustrates the SNMP configuration required to enable traps when interfaces change between up and down states.

Click here to view code image

```
snmp-server enable traps snmp linkup linkdown
```

The following example shows a SNMP debug message generated when a physical interface changes state.

Click here to view code image

```
SNMP: V1 Trap, ent snmpTraps, addr 10.10.10.1, gentrap 2, spectrap 0
 ifIndex.1 = 1
 ifDescr.1 = Ethernet0/0
 ifType.1 = 6
 lifEntry.20.1 = administratively down
```

In most situations, the transport network will require a routing protocol to operate. If this is the case, then the neighbor adjacency should be monitored to ensure that there is no loss of connectivity with the routing peer, which results in the loss of routes for the IPsec peer.

The following example illustrates EIGRP and BGP logging messages that are generated when a neighbor is removed.

Click here to view code image

```
%DUAL-5-NBRCHANGE: IP-EIGRP 109: Neighbor 10.1.1.1 (GigabitEthernet1/2) is
down: peer restarted
%DUAL-5-NBRCHANGE: IP-EIGRP 109: Neighbor 10.1.1.1 (GigabitEthernet1/2) is
down: interface down
%DUAL-5-NBRCHANGE: IP-EIGRP 109: Neighbor 10.1.1.1 (GigabitEthernet1/2) is up:
new adjacency

%BGP-5-ADJCHANGE: neighbor 10.1.1.1 Up
%BGP-5-ADJCHANGE: neighbor 10.1.1.1 Down BGP Notification sent
%BGP-5-ADJCHANGE: neighbor 10.1.1.1 Up
```

SNMP can be used to indicate routing neighborships changing. The example below shows the configuration to enable traps for EIGRP, and it also illustrates the SNMP message that is generated when a EIGRP neighbor changes.

Click here to view code image

```
snmp-server enable traps eigrp
SNMP can be used to alert on
SNMP: Queuing packet to 10.10.10.1
SNMP: V1 Trap, ent ciscoEigrpMIB, addr 10.10.10.1, gentrap 6, spectrap 3
 cEigrpPeerAddr.0.1.0 = 10.10.50.8
 cEigrpPeerAddrType.0.1.0 = 1
```

Other routing protocols, such as BGP or OSPF, can be configured using SNMP to alert when connectivity changes. These are configured using the **snmp-server enable traps** command, similar to the command in the previous example; however, their configuration will use the **bgp** or **ospf** keywords instead of **eigrp**.

Knowing that a routing peer was lost does not help when a device loses reachability for the prefix that contains the tunnel destination. In this scenario, dead peer detection will activate, bringing the IKE session down and the corresponding tunnel into the down state.

Should a routing protocol be running over the IPsec Security Association, the same method can be used for monitoring. However, it should be noted that in this case the overlay network is being monitored and not the transport network.

VPN Tunnel Establishment

When an IKEv2 session is successfully created, SNMP or syslog messages can be used to send an alert.

Cisco IPsec Flow Monitor MIB

When SNMP is used to monitor IKE or IPsec, it is advisable to use the Cisco-IPsec-Flow-Monitor-MIB. This is based on the IETF draft-ietf-IPsec-flow-monitoring-mib-01 but is updated to reflect IKEv2. For monitoring, the MIB consists of notifications types, which contains objects. The updates consist of new objects, such as email address or Distinguished Name (DN) for the IKE ID, which is described in the notification cikeTunLocalType. The cikeNegMode notification, which in IKEv1 previously contained objects for aggressive mode and main mode, has been updated to include an object for IKEv2.

This is generally considered to be the de-facto MIB when VPNs are used. The MIB is structured for monitoring, trending, and troubleshooting IPsec connections. Data gained by this MIB can be used to identify trends and failure events. The MIB is split into two components: IKE (versions 1 and 2) is covered with cike, and IPsec with cipsec.

The MIB was designed based on specific requirements from service providers who wanted to offer an outsourced VPN service to customers. The main focus was the provision of services in such a way that would satisfy a Service Level Agreement (SLA).

SNMP with IKEv2

SNMP traps can be configured to alert when an IKE session starts with the command shown in the following example.

Click here to view code image

```
snmp-server enable traps ike tunnel start
```

This will generate a notification that contains the local and remote addresses and the lifetime of the tunnel. The following example illustrates a message generated when the IKE session is started. The local address is shown as the IKE identity, along with the IP address, which is shown as hex 0x0A0A0A02, which is 10.10.10.2 in decimal.

Click here to view code image

```
SNMPv2-MIB::snmpTrapOID.0 = OID: CISCO-IPSEC-FLOW-MONITOR-MIB::cikeTunnelStart
CISCO-IPSEC-FLOW-MONITOR-MIB::cikePeerLocalAddr.9."hostname=router1.cisco.com".9
"hostname=router2.cisco.com".9 = Hex-STRING: 0A 0A 0A 02
CISCO-IPSEC-FLOW-MONITOR-MIB::cikePeerRemoteAddr.9."hostname=router1.cisco.com".9
"hostname=router2.cisco.com"9 = Hex-STRING: 0A 0A 0A 01

CISCO-IPSEC-FLOW-MONITOR-MIB::cikeTunLifeTime.9 = INTEGER: 86400 seconds
```

SNMP notifications can be enabled when an IKE session is removed. This is achieved using the command shown in the following example. The notification ikeTunnelStop contains objects to describe the local and remote IKE identities, the termination reason, and the lifetime of the IKE SA.

Click here to view code image

```
snmp-server enable traps ike tunnel stop
```

This will generate a notification that contains the local and remote addresses, the reason the IKE session was removed, and the lifetime of the IKE SA.

The following example illustrates a SNMP notification generated when an IKE session is deleted. The local address is shown as the IKE identity, along with the IP address. The tunnel active time shows the lifetime of the tunnel and the reason for deletion.

Click here to view code image

```
SNMPv2-MIB::snmpTrapOID.0 = OID: CISCO-IPSEC-FLOW-MONITOR-MIB::cikeTunnelStop
CISCO-IPSEC-FLOW-MONITOR-MIB::cikePeerLocalAddr.9."hostname=router1.cisco.com".9
"hostname=router2.cisco.com".9 = Hex-STRING: 0A 0A 0A 02
CISCO-IPSEC-FLOW-MONITOR-MIB::cikePeerRemoteAddr.9."hostname=router1.cisco.com".9
"hostname=router2.cisco.com"9 = Hex-STRING: 0A 0A 0A 01


CISCO-IPSEC-FLOW-MONITOR-MIB::cikeTunActiveTime.3 = INTEGER: 11
CISCO-IPSEC-FLOW-MONITOR-MIB::cikeTunHistTermReason.3 = INTEGER: other(1)
```

Details of IKE sessions are stored within the MIB and can be obtained by using SNMP or locally by using the **show crypto mib ike flowmib** command, as the following example illustrates.

Click here to view code image

```
Router#show crypto mib ike flowmib ?
  failure  Shows ike mib failure table
  global   Shows ike mib global statistics
  history  Shows ike mib history table
  peer     Shows ike mib peer info
  tunnel   Shows ike mib tunnel info
```

For previous sessions the history keyword can be used to show statistics such as the transmitted octets and number of packets, along with the authentication method and cryptographic algorithms used.

The global keyword can be used to display statistics pertaining to global IKE sessions that have occurred on the device.

By default the details of 200 sessions are stored locally. This value can be changed with the command illustrated in the following example.

Click here to view code image

```
crypto mib ipsec flow history tunnel size <2-200>
```

The following example illustrates the IKE SNMP objects within the MIB.

Click here to view code image

```
Router#show snmp mib | i cike
cikeGlobalActiveTunnels
cikeGlobalPreviousTunnels
cikeGlobalInOctets
cikeGlobalInPkts
cikeGlobalInDropPkts
cikeGlobalInNotifys
cikeGlobalInP2Exchgs
cikeGlobalInP2ExchgInvalids
cikeGlobalInP2ExchgRejects
cikeGlobalInP2SaDelRequests
```

Table 13-3 lists the SNMP trap commands that enable message generation when an IKE SA is created or terminated. The notification type is listed, along with the objects contained within this.

Command	Notification	Objects
snmp-server enable traps ike tunnel start	cikeTunnelStart	cikePeerLocalAddr cikePeerRemoteAddr cikeTunLifeTime
snmp-server enable traps ike tunnel stop	cikeTunnelStop	cikeTunHistTermReason cikePeerLocalAddr cikePeerRemoteAddr cikeTunActiveTime

Table 13-3 *SNMP IKE Trap Commands*

Note that with SNMP statistics for IKE, these are global for both IKEv1 and IKEv2. There is no differentiation between the values for each protocol.

Syslog

It is possible to activate syslog messages to report on events that are related to the creation and deletion of IKEv2 SAs. The following example illustrates the command to enable the generation of syslog messages in relation to IKEv2.

Click here to view code image

```
Router(config)#crypto logging ikev2
```

If syslog messages are enabled, when a IKEv2 SA is created the message shown in the following example is generated.

```
%IKEV2-5-SA_UP: SA UP
```

When syslog messages are enabled, when an IKEv2 SA is deleted the message shown in the following example is generated.

```
%IKEV2-5-SA_DOWN: SA DOWN
```

It is possible to enable syslog messages for the creation or deletion of a crypto session. The following example illustrates enabling the generation of crypto session messages.

Click here to view code image

```
Router(config)#crypto logging session
```

The following example illustrates the message generated when an IKEv2 session is created.

Click here to view code image

```
%CRYPTO-5-IKEV2_SESSION_STATUS: Crypto tunnel v2 is UP.  Peer 10.10.10.1:500
Id: hostname=Router.cisco.com
```

The following example illustrates the message generated when an IKEv2 session is deleted.

Click here to view code image

```
%CRYPTO-5-IKEV2_SESSION_STATUS: Crypto tunnel v2 is DOWN.  Peer 10.10.10.1:500
Id: hostname=Router1.cisco.com
```

Most syslog messages related to IKEv2 are generated with a severity level of 5, which indicates a notification; however, if a negotiation aborted message is generated, the severity level is 3. When a cookie challenge is sent, the severity level is 1. Table 13-4 illustrates the IKEv2 messages and the corresponding syslog severity level. Having a minimum severity of 5 requires the logging level to be set to 5 or above for the messages to be generated and seen.

IKEv2 Message	Syslog Severity Level
OSAL_INITIATE_TUNNEL	Notify (5)
RECV_CONNECTION_REQUEST	Notify (5)
IPSEC_COLLISION, IKEV2	Notify (5)
SA_UP, IKEV2, LOG_NOTICE	Notify (5)
SA_DOWN, IKEV2, LOG_NOTICE	Notify (5)
SA_REJECTED_RESOURCE_LOW	Notify (5)
SA_REJECTED_LIMIT_REACHED	Notify (5)
INCREASE_PEER_WINDOW_SIZE	Notify (5)
NEG_ABORT	Error (3)
SEND_COOKIE_CHALLENGE	Alert (1)

Table 13-4 *IKEv2 Syslog Message Severity Level*

When IKEv2 syslog logging is enabled, if an IKE session fails to be established, a syslog message will be generated. The following example illustrates a typical message that is generated; however, the exact message will vary depending on the nature of the cause. This will require further examination to determine the specific peer and the reason for the authentication failure.

Click here to view code image

```
%IKEV2-3-NEG_ABORT: Negotiation aborted due to ERROR: Auth exchange failed

%IKEV2-3-NEG_ABORT: Negotiation aborted due to ERROR: Received no proposal chosen
notify
```

Pre-Shared Key Authentication

AAA accounting allows for session information to be generated, indicating when the session starts and when it stops. In addition, session identity information and session usage information will be passed to the RADIUS server via RADIUS attributes and VSA (vendor-specific attributes). RADIUS accounting is used to record user activity for auditing purposes in many large-scale networks, and its use is suitable when a historic record of VPN sessions is required.

When PSK is used as the authentication mechanism, accounting can be enabled to report on successful authentication and creation of the IPsec Security Association. On creation of this event, IKE makes a call to AAA to start an accounting record. This request is sent to the AAA server and contains the

calling-station-id, which is the source IP address of the peer. The user-name is the IKE identity. The following example illustrates the accounting data generated when an IKE session is created.

Click here to view code image

```
AAA/ACCT/EVENT/(00000011): CALL START
AAA/ACCT/EVENT/(00000011): IPSEC TNL UP
AAA/ACCT/IPSEC-TUNNEL(00000011): Queueing record is START

IKEv2:(SA ID = 1):[IKEv2 -> AAA] Accounting start request sent successfully
RADIUS(00000011): Send Accounting-Request to 172.16.1.3:1646 id 1646/9, len 256
RADIUS:  authenticator 6A 7A 8F EA 4E 10 E9 CE - 8C C9 EC A8 C4 70 C7 9F
RADIUS:  Acct-Session-Id     [44]  10  "00000007"
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  60
RADIUS:   Cisco AVpair       [1]   54  "audit-session-id=L2L4AC1G102ZO2L4AC1G101Z
I1F401F4ZO8"
RADIUS:  Vendor, Cisco       [26]  40
RADIUS:   Cisco AVpair       [1]   34  "isakmp-phase1-id=Router1.Domain1"
RADIUS:  Vendor, Cisco       [26]  37
RADIUS:   Cisco AVpair       [1]   31  "isakmp-initator-ip=172.16.1.1"
RADIUS:  User-Name           [1]   17  "Router1.Domain1"
RADIUS:  Vendor, Cisco       [26]  36
RADIUS:   Cisco AVpair       [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Authentic      [45]  6   Local                     [2]
RADIUS:  Acct-Status-Type    [40]  6   Start                     [1]
RADIUS:  NAS-IP-Address      [4]   6   172.16.1.2
RADIUS:  Acct-Delay-Time     [41]  6   0
```

The accounting stop record is sent after the IKE/IPsec session is deleted. The accounting stop record includes the calling-station-id, which contains the peer IP address, and the username-name, which contains the peer IKE identity. The lifetime of the session is included, and the amount of incoming and outgoing octets and packets. This information can be used to determine how long a session lasted for and how much data was transmitted.

There was a customer case where an IPsec profile was configured to use Perfect Forward Secrecy (PFS); however certain clients did not support PFS and would fail rekey. Users reported a degraded service. When the AAA accounting data was examined, it became apparent that the lifetime of sessions that users reported as degraded would only last a maximum of one hour before PFS would be attempted when a rekey occurred, at this point the rekey would fail and the session be torn down.

The following example illustrates the accounting data generated when a IKE session is torn down.

Click here to view code image

```
AAA/ACCT/EVENT/(00000011): IPSEC TNL DOWN
IKEv2:in_octets 13020, out_octets 17640
IKEv2:in_packets 105, out_packets 105
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Accounting stop request sent successfully
AAA/ACCT/EVENT/(00000011): CALL STOP
AAA/ACCT/IPSEC-TUNNEL(00000011): STOP

RADIUS(00000011): Send Accounting-Request to 172.16.1.3:1646 id 1646/10, len 324
RADIUS:  authenticator 6E 18 B4 4F E0 B5 C0 09 - 9F B9 FC 01 5A D4 69 6F
RADIUS:  Acct-Session-Id     [44]  10  "00000007"
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  60
RADIUS:   Cisco AVpair       [1]   54  "audit-session-id=L2L4AC1G102ZO2L4AC1G101Z
I1F401F4ZO8"
```

```
RADIUS:  Vendor, Cisco        [26]  40
RADIUS:   Cisco AVpair        [1]   34  "isakmp-phase1-id=Router1.Domain1"
RADIUS:  Vendor, Cisco        [26]  37
RADIUS:   Cisco AVpair        [1]   31  "isakmp-initator-ip=172.16.1.1"
RADIUS:  User-Name            [1]   17  "Router1.Domain1"
RADIUS:  Acct-Authentic       [45]  6   Local                     [2]
RADIUS:  Vendor, Cisco        [26]  36
RADIUS:   Cisco AVpair        [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Session-Time    [46]  6   34
RADIUS:  Acct-Input-Octets    [42]  6   13020
RADIUS:  Acct-Output-Octets   [43]  6   17640
RADIUS:  Acct-Input-Packets   [47]  6   105
RADIUS:  Acct-Output-Packets  [48]  6   105
RADIUS:  Acct-Terminate-Cause[49]  6   none                      [0]
RADIUS:  Vendor, Cisco        [26]  32
RADIUS:   Cisco AVpair        [1]   26  "disc-cause-ext=No Reason"
RADIUS:  Acct-Status-Type     [40]  6   Stop                      [2]
RADIUS:  NAS-IP-Address       [4]   6   172.16.1.2
RADIUS:  Acct-Delay-Time      [41]  6   0
```

PKI Authentication

When authentication occurs using PKI, the PKI subsystem is generally called by the IKE subsystem, and the IKE subsystem will generally generate informational messages.

There are a number of syslog messages generated by PKI when error conditions occur. These messages pertain to issues encountered with certificates, such as the peer certificate not being within the validity period, revoked, or expired.

In many instances the cause of PKI failures is an incorrect time setting, which results in certificate validity errors being generated.

There are a number of syslog messages generated by the PKI subsystem. The following example illustrates some of the syslog PKI messages that can be generated that contain certificate-related conditions.

The following lists illustrate error messages that are generated for PKI-related events: specifically, certificate error messages, CA-related error messages, and renewal error messages.

- PKI certificate error messages
  - %PKI-3-CERTIFICATE_INVALID:
  - %PKI-3-CERTIFICATE_INVALID_EXPIRED:
  - %PKI-3-CERTIFICATE_INVALID_NOT_YET_VALID:
  - %PKI-3-CERTIFICATE_REVOKED:
- PKI CA-related error messages
  - %PKI-3-GETCARACERT:
  - %PKI-3-INVALIDCACERT:
  - %PKI-3-POLLCACERT:
  - %PKI-3-POLLROUTERCERT:
  - %PKI-3-SOCKETSELECT:
  - %PKI-3-SOCKETSEND:
- PKI certificate renewal error messages

- %PKI-4-CRL_LDAP_QUERY:
- %PKI-4-CS_PUBLISH_STORAGE:
- %PKI-4-NOAUTOSAVE:
- %PKI-4-NOCONFIGAUTOSAVE:
- %PKI-4-NOSHADOWAUTOSAVE:
- %PKI-6-AUTOENROLL_KEY_LOCKED:
- %PKI-6-AUTOSAVE:
- %PKI-6-CERTFAIL:
- %PKI-6-CERT_FATAL_ERR:
- %PKI-6-CERTIFSRECV:
- %PKI-6-CERTIFSSEND:
- %PKI-6-CERTPENDING:
- %PKI-6-CERTREJECT:
- %PKI-6-CERTRET:
- %PKI-6-CONFIGAUTOSAVE:

AAA accounting can be enabled for PKI authentication. The following example illustrates the configuration used to create the accounting information.

Click here to view code image

```
aaa new-model

aaa group server radius RA-AUTHC-SG-1
 server-private 172.16.100.100 auth-port 1812 key Cisco123

aaa accounting network RA-ACCC-LIST-1 start-stop group RA-AUTHC-SG-1

crypto ikev2 profile ALL-SPOKES
 match certificate certmap
 identity local dn
 authentication remote rsa-sig
 authentication local rsa-sig
 pki trustpoint MYCERT
 aaa accounting cert RA-ACCC-LIST-1
 virtual-template 1
```

The following example shows an accounting request that was generated for an incoming successful authentication. The IKE identity can be seen in addition to the source IP address, which will be the post-NAT address of the peer (if NAT was used on the transport network).

Click here to view code image

```
RADIUS(00000016): Send Accounting-Request to 172.16.100.100:1646 id 1646/17,
len 284
RADIUS:  authenticator 1D C9 AB 28 97 97 98 D7 - 01 F1 35 3D 79 F1 53 16
RADIUS:  Acct-Session-Id     [44]  10  "0000000A"
RADIUS:  Calling-Station-Id  [31]  11  "209.1.1.2"
RADIUS:  Vendor, Cisco       [26]  62
RADIUS:   Cisco AVpair       [1]   56  "audit-session-id=L2L4C8010102ZO2L4D101010
2ZI1F401F4ZOE"
```

```
RADIUS:  Vendor, Cisco        [26]  54
RADIUS:   Cisco AVpair        [1]   48  "isakmp-phase1-id=hostname=Spoke-1.cisco
.com"
RADIUS:  Vendor, Cisco        [26]  36
RADIUS:   Cisco AVpair        [1]   30  "isakmp-initator-ip=209.1.1.2"
RADIUS:  User-Name            [1]   31  "hostname=Spoke-1.cisco.com"
RADIUS:  Vendor, Cisco        [26]  36
RADIUS:   Cisco AVpair        [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Authentic       [45]  6   Local                     [2]
RADIUS:  Acct-Status-Type     [40]  6   Start                     [1]
Hub-1(config-ikev2-profile)#
RADIUS:  NAS-IP-Address       [4]   6   172.16.0.1
RADIUS:  Acct-Delay-Time      [41]  6   0
RADIUS(00000016): Sending a IPv4 Radius Packet
RADIUS(00000016): Started 5 sec timeout
```

The following example shows an accounting request that was generated for the termination of a PKI-authenticated session The IKE identity can be seen, in addition to the source IP address (post-NAT if NAT was used) of the peer and the traffic sent and session duration.

Click here to view code image

```
RADIUS(00000016): Send Accounting-Request to 172.16.100.100:1646 id 1646/26,
len 352
RADIUS:  authenticator D2 06 49 A8 79 F8 77 04 - E3 58 B9 4F D5 66 5E E8
RADIUS:  Acct-Session-Id      [44]  10  "0000000A"
RADIUS:  Calling-Station-Id   [31]  11  "209.1.1.2"
RADIUS:  Vendor, Cisco        [26]  62
RADIUS:   Cisco AVpair        [1]   56  "audit-session-id=L2L4C8010102ZO2L4D101010
2ZI1F401F4ZOE"
RADIUS:  Vendor, Cisco        [26]  54
RADIUS:   Cisco AVpair        [1]   48  "isakmp-phase1-id=hostname=Spoke-1.cisco
.com"
RADIUS:  Vendor, Cisco        [26]  36
RADIUS:   Cisco AVpair        [1]   30  "isakmp-initator-ip=209.1.1.2"
RADIUS:  User-Name            [1]   31  "hostname=Spoke-1.cisco.com"
RADIUS:  Acct-Authentic       [45]  6   Local                     [2]
RADIUS:  Vendor, Cisco        [26]  36
RADIUS:   Cisco AVpair        [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Session-Time    [46]  6   39
RADIUS:  Acct-Input-Octets    [42]  6   15222
RADIUS:  Acct-Output-Octets   [43]  6   15641
RADIUS:  Acct-Input-Packets   [47]  6   22
RADIUS:  Acct-Output-Packets  [48]  6   17
RADIUS:  Acct-Terminate-Cause[49]  6   none                      [0]
RADIUS:  Vendor, Cisco        [26]  32
RADIUS:   Cisco AVpair        [1]   26  "disc-cause-ext=No Reason"
RADIUS:  Acct-Status-Type     [40]  6   Stop                      [2]
RADIUS:  NAS-IP-Address       [4]   6   172.16.0.1
RADIUS:  Acct-Delay-Time      [41]  6   0
RADIUS(00000016): Sending a IPv4 Radius Packet
```

EAP Authentication

Like the other authentication methods, EAP can generate accounting data when a session is successfully created or destroyed.

The following example illustrates the configuration used to create the accounting information.

Click here to view code image

```
aaa new-model

aaa group server radius RA-AUTHC-SG-1
 server-private 172.16.100.100 auth-port 1812 key Cisco123

aaa authentication login RA-AUTHC-LIST-1 group RA-AUTHC-SG-1
aaa authorization network RA-AUTHZ-LIST-1 local
aaa accounting network RA-ACCC-LIST-1 start-stop group RA-AUTHC-SG-1

crypto ikev2 profile RA-CLIENT
 match identity remote address 0.0.0.0
 identity local dn
 authentication remote eap query-identity
 authentication local rsa-sig
 pki trustpoint MYCERT
 aaa authentication eap RA-AUTHC-LIST-1
 aaa authorization group eap list RA-AUTHZ-LIST-1 RA-LOCAL-POLICY-1
 aaa accounting eap RA-ACCC-LIST-1
 virtual-template 10
```

The same attributes, such as Calling-Station-Id, are used to record the source IP address used, along with the IKE identity, which is defined using isakmp-phase1-id, and the isakmp-initator-ip value is the source IP address. The username presented can be seen by the value user-name, and the framed-ip-address shows the IP address allocated via the local IP pool.

The following example illustrates the accounting data generated when an IKE session is created.

Click here to view code image

```
RADIUS/ENCODE: Best Local IP-Address 172.16.0.1 for Radius-Server 172.16.100.100
RADIUS(00000014): Send Accounting-Request to 172.16.100.100:1646 id 1646/1, len
256
RADIUS:  authenticator 50 3B BC 65 EB 53 6A 91 - 44 13 F0 FE 6D 06 01 2F
RADIUS:  Calling-Station-Id  [31]  14  "209.1.10.100"
RADIUS:  Vendor, Cisco       [26]  63
RADIUS:   Cisco AVpair       [1]   57  "audit-session-id=L2L4C8010102ZO2L4D1010A6
4ZH11941194ZO6"
RADIUS:  Vendor, Cisco       [26]  37
RADIUS:   Cisco AVpair       [1]   31  "isakmp-phase1-id=209.1.10.100"
RADIUS:  Vendor, Cisco       [26]  39
RADIUS:   Cisco AVpair       [1]   33  "isakmp-initator-ip=209.1.10.100"
RADIUS:  Framed-IP-Address   [8]   6   172.16.101.10
RADIUS:  Acct-Session-Id     [44]  10  "00000006"
RADIUS:  User-Name           [1]   7   "user1"
RADIUS:  Vendor, Cisco       [26]  36
Hub-1#
RADIUS:   Cisco AVpair       [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Authentic      [45]  6   Local                    [2]
RADIUS:  Acct-Status-Type    [40]  6   Start                    [1]
RADIUS:  NAS-IP-Address      [4]   6   172.16.0.1
RADIUS:  Acct-Delay-Time     [41]  6   0
```

The following example illustrates the accounting data generated when an IKE session is torn down. The information about the session can be seen, with User-Name denoting the EAP username used to identify the session.

Click here to view code image

```
RADIUS(00000014): Send Accounting-Request to 172.16.100.100:1646 id 1646/6, len 324
RADIUS:  authenticator B5 59 09 6E F0 A5 8D F0 - 37 1B 64 53 C5 AC 00 91
```

```
RADIUS:  Calling-Station-Id  [31]  14  "209.1.10.100"
RADIUS:  Vendor, Cisco       [26]  63
RADIUS:   Cisco AVpair       [1]   57  "audit-session-id=L2L4C8010102ZO2L4D1010A6
4ZH11941194ZO6"
RADIUS:  Vendor, Cisco       [26]  37
RADIUS:   Cisco AVpair       [1]   31  "isakmp-phase1-id=209.1.10.100"
RADIUS:  Vendor, Cisco       [26]  39
RADIUS:   Cisco AVpair       [1]   33  "isakmp-initator-ip=209.1.10.100"
RADIUS:  Framed-IP-Address   [8]   6   172.16.101.10
RADIUS:  Acct-Session-Id     [44]  10  "00000006"
RADIUS:  User-Name           [1]   7   "user1"
RADIUS:  Acct-Authentic      [45]  6   Local                    [2]
RADIUS:  Vendor, Cisco       [26]  36
RADIUS:   Cisco AVpair       [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Session-Time   [46]  6   51
RADIUS:  Acct-Input-Octets   [42]  6   7577
RADIUS:  Acct-Output-Octets  [43]  6   3064
RADIUS:  Acct-Input-Packets  [47]  6   91
RADIUS:  Acct-Output-Packets [48]  6   23
RADIUS:  Acct-Terminate-Cause[49]  6   none                     [0]
RADIUS:  Vendor, Cisco       [26]  32
RADIUS:   Cisco AVpair       [1]   26  "disc-cause-ext=No Reason"
RADIUS:  Acct-Status-Type    [40]  6   Stop                     [2]
RADIUS:  NAS-IP-Address      [4]   6   172.16.0.1
RADIUS:  Acct-Delay-Time     [41]  6   0
```

Authorization Using RADIUS-Based AAA

RADIUS combines authentication and authorization into a single function. The Access-Accept packet sent by the RADIUS server to the client contains authorization information, which may create difficultly when attempting to decouple authentication and authorization.

The authorization policy used can be determined from enabling debugging on the VPN gateway itself. This does not scale for large-scale deployments, so it is more prudent to enable this only when it is specifically required.

The following example illustrates the logging messages pertaining to authorization from IKE and AAA. These were enabled using the **debug crypto ikev2** and **debug aaa authorization** commands.

Click here to view code image

```
IKEv2:Using mlist default and username service1 for group author request
AAA/BIND(0000004E): Bind i/f
AAA/AUTHOR (0x4E): Pick method list 'default'
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorisation request sent
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response
```

When RADIUS is used, an alternative to enabling debugging is to obtain data pertaining to authorization events by viewing the logs on the RADIUS server itself. The requested authorization policy can usually be viewed in a log message that contains the Access-Accept information.

Data Encryption: SNMP with IPsec

SNMP can be used for generating notifications when an IPsec Security Association is created or destroyed. The following example illustrates the command that enables traps to be generated when an IPsec tunnel is created.

Click here to view code image

```
snmp-server enable traps ipsec tunnel start
```

The following example illustrates the command that enables traps to be generated when an IPsec tunnel is destroyed.

Click here to view code image

```
snmp-server enable traps ipsec tunnel start
```

The IPsec SNMP trap contains a great deal of information, including the tunnel source and destination IP addresses, lifetime, tunnel protocol, and the reason for the tunnel's termination. The following example illustrates a trap when an IPsec tunnel is removed; the source and destination IP addresses are seen in hexadecimal (0A 0A 0A 01 representing 10.10.10.1 in decimal). The protocol is 47, indicating GRE. The tunnel is being terminated because the peer was lost.

Click here to view code image

```
SNMPv2-MIB::snmpTrapOID.0 = OID: CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecTunnelStop

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecTunActiveTime.13 = INTEGER: 301032

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecTunHistActiveTime.13 = INTEGER: 301032

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecTunHistLifeTime.13 = INTEGER: 3600 Seconds

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecTunHistLifeSize.13 = INTEGER: 4608000 KBytes

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecTunHistTermReason.13 = INTEGER: peerLost (5)

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistLocalType.13 = INTEGER:
singleIpAddr(1)

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistRemoteType.13 = INTEGER:
singleIpAddr(1)

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistLocalAddr1.13 = Hex-STRING: 0A 0A
0A 01

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistLocalAddr2.13 = Hex-STRING: 0A 0A
0A 01

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistRemoteAddr1.13 = Hex-STRING: 0A 0A
0A 02

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistRemoteAddr2.13 = Hex-STRING: 0A 0A
0A 02

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistLocalProtocol.13 = INTEGER: 47

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistLocalPort.13 = INTEGER: 0
```

Similar to IKE, the SNMP information is stored within the MIB. The following example illustrates how to view the SNMP objects.

Click here to view code image

```
Router#show snmp mib | i cipSec
cipSecMibLevel
cipSecGlobalActiveTunnels
cipSecGlobalPreviousTunnels
```

```
cipSecGlobalInOctets
cipSecGlobalHcInOctets
cipSecGlobalInOctWraps
cipSecGlobalInDecompOctets
cipSecGlobalHcInDecompOctets
cipSecGlobalInDecompOctWraps
```

Table 13-5 illustrates the SNMP trap commands used to enable message generation when an IPsec Security Association is created. The notification type is listed, along with the objects contained within the notification.

Command	Notification	Objects
snmp-server enable traps IPsec tunnel start	cipSecTunnelStart	cipSecTunLifeTime
		cipSecTunLifeSize
		cipSecEndPtLocalType
		cipSecEndPtLocalAddr1
		cipSecEndPtLocalAddr2
		cipSecEndPtRemoteType
		cipSecEndPtRemoteAddr1
		cipSecEndPtRemoteAddr2
		cipSecEndPtLocalProtocol
		cipSecEndPtLocalPort
snmp-server enable traps ipsec tunnel stop	cipSecTunnelStop	cipSecTunActiveTime
		cipSecTunHistActiveTime
		cipSecTunHistLifeTime
		cipSecTunHistLifeSize
		cipSecTunHistTermReason
		cipSecEndPtHistLocalType
		cipSecEndPtHistRemoteType
		cipSecEndPtHistLocalAddr1
		cipSecEndPtHistLocalAddr2
		cipSecEndPtHistRemoteAddr1
		cipSecEndPtHistRemoteAddr2
		cipSecEndPtHistLocalProtocol
		cipSecEndPtHistLocalPort

Table 13-5 *SNMP IPsec Trap Commands*

Overlay Routing

If a routing protocol is being used over the created IPsec Security Association, this should be monitored. The best tools to monitor a routing protocol are either SNMP or syslog, both of which were covered within the earlier section on IP connectivity. The same concepts can be used again for overlay routing. If logs for both overlay and transport routing are used, care should be taken to ensure that there is a distinct differentiation between them.

When IKEv2 routing is used, specific focus can be used to monitor the addition and deletion of virtual-access interfaces. The following example illustrates the command that enables traps when a virtual-access interface is created.

Click here to view code image

```
snmp-server enable traps snmp linkdown linkup
```

The following example illustrates the SNMP trap generated when a virtual-access interface is created and removed. The virtual-access interface number can be seen, as well as the state of the tunnel (Tunnel Up or Tunnel Down).

Click here to view code image

```
ifIndex.24 = 24
 ifDescr.24 = Virtual-Access1
 ifType.24 = 131
 lifEntry.20.24 = Tunnel Up

ifIndex.24 = 24
 ifDescr.24 = Virtual-Access1
 ifType.24 = 131
 lifEntry.20.24 = Tunnel Down
```

If IKEv2 routing is used, when information about IP addresses is sent via a configuration payload, the local Routing Information Base (RIB) is updated. If this information is required to enable generation of information that pertains to local static routing, logging can be enabled by turning on debugging for static routing. The following example illustrates enabling static routing debugging. IKEv2 logging was also enabled to show the session being created and then the relevant IP address being assigned.

Click here to view code image

```
Router#debug ip routing static detail

%CRYPTO-5-IKEV2_SESSION_STATUS: Crypto tunnel v2 is UP.  Peer 10.10.60.6:500
f_vrf:  OUTSIDE i_vrf:  OUTSIDE   Id: hostname=Router.cisco.com
%LINEPROTO-5-UPDOWN: Line protocol on Interface Virtual-Access3, changed state to
up

IP-ST(default): updating same distance on 192.168.2.37/32
IP-ST(default):  192.168.2.37/32 [1], Virtual-Access3 Path =, add succeed, active
state
```

Manual verification can ensure that the IP address that was assigned to the peer is reachable via the virtual-access interface.

Click here to view code image

```
Router#show ip route 192.168.2.36
Routing entry for 192.168.2.36/32
  Known via "static", distance 1, metric 0 (connected)
```

```
Tag 1
Routing Descriptor Blocks:
* directly connected, via Virtual-Access1, permanent
    Route metric is 0, traffic share count is 1
    Route tag 1
```

Data Usage

Statistics on the amount of data sent over a tunnel interface can be obtained from SNMP or AAA accounting. Knowing what sort of data is traveling within the tunnel can be useful for policy or billing purposes. For this requirement, NetFlow can be used to record all flows that have occurred over the tunnel interface. Tunnel interfaces or virtual-access interfaces can have NetFlow configuration applied to them to record details of flows prior to encapsulation and encryption.

There are a number of reasons why details about flows are needed. NetFlow provides a very lightweight form of network monitoring, in which network traffic can be visualized in almost real time. Applications can be monitored, along with profiles, which allows for the allocation of capacity and the monitoring of usage policy . Information from all flows traversing an IPsec Security Association can be stored; these can be warehoused for later analysis via data mining or similar processes for marketing or security purposes.

The following example illustrates the configuration that enables NetFlow data to be gathered from a virtual-access interface that is spawned from a virtual-template interface.

The exporter is configured, which is where the NetFlow data will be sent. The export protocol is configured as ipfix; NetFlow version 9 or 5 can also be configured.

```
flow exporter exp-1
 destination 10.10.10.100
 export-protocol ipfix
```

A flow record is created (using the **match** command), with details of the source and destination IP addresses, protocols, ports, and TOS value. Each of these attributes is matched to create a unique record, with the total bytes and packets sent for the flow being collected (using the **collect** command).

Click here to view code image

```
flow record v4_r1
match ipv4 tos
match ipv4 protocol
match ipv4 source address
match ipv4 destination address
match transport source-port
match transport destination-port
collect counter bytes long
collect counter packets long
```

A flow monitor is created, which uses the record created earlier and exports this to the exporter configured earlier.

```
flow monitor f-monitor-1
 record v4_r1
 exporter exp-1
 exit
```

The flow monitor is applied to the virtual-template interface. This will then record any outgoing or

incoming flows over the IPsec Security Associations associated with the cloned virtual-access interfaces.

Click here to view code image

```
interface Virtual-Template2 type tunnel
 ip unnumbered Loopback2
 ip flow monitor f-monitor-1 input
 ip flow monitor f-monitor-1 output
```

When a virtual-access interface is spawned from the virtual-template, it will inherit the relevant configuration. The following example illustrates the virtual-access interface with the configuration applied.

Click here to view code image

```
Router#show derived-config interface virtual-access 2
Building configuration...

Derived configuration : 421 bytes

interface Virtual-Access2
 ip unnumbered Loopback2
 ip flow monitor f-monitor-1 input
 ip flow monitor f-monitor-1 output
```

Any traffic flows traversing the tunnel interface will be sent to the NetFlow exporter. The following example illustrates a flow recorded in the cache; the source and destination IP addresses can be seen, along with the ports. Counters relating to the flow are also recorded.

Click here to view code image

```
Router#show flow monitor f-monitor-1 cache
  Cache type:                               Normal
  Cache size:                                 4096
  Current entries:                               1
  High Watermark:                                3

  Flows added:                                   8
  Flows aged:                                    7
    - Active timeout      (  1800 secs)          0
    - Inactive timeout    (    15 secs)          7
    - Event aged                                 0
    - Watermark aged                             0
    - Emergency aged                             0

IPV4 SOURCE ADDRESS:       192.168.2.4
IPV4 DESTINATION ADDRESS:  192.168.2.1
TRNS SOURCE PORT:          38658
TRNS DESTINATION PORT:     23
IP TOS:                    0xC0
IP PROTOCOL:               6
counter bytes long:        1545
counter packets long:      37

Router#show flow exporter statistics
Flow Exporter exp-1:
  Packet send statistics (last cleared 00:20:18 ago):
    Successfully sent:         4                     (438 bytes)
```

```
Client send statistics:
  Client: Flow Monitor f-monitor-1
    Records added:           3
      - sent:                3
    Bytes added:             90
      - sent:                90
```

Note

The preceding examples all used IPv4; it is also possible to use IPv6 flows if they are required.

Summary

Because many components are used to construct an IPsec VPN, there are many components that can be monitored for errors or abnormal conditions.

An IPsec Security Association that uses IKEv2 will be constructed using a number of protocols and methods. Each component can be monitored in a layered fashion, just as the protocols operate in a layered fashion.

Protocols such as SNMP, netflow, and syslog can be used to alert on event pertaining to the creation or deletion of an IKE or IPsec Security Association. RADIUS and AAA events can be used to provide alerts when AAA conditions occur.

Care should be taken when any monitoring solution is employed that uses debug statements, for these generally create a very large amount of data and can put pressure on other resources; in addition, they can generate a vast amount of noise. Monitoring should be designed so that only critical events cause alerts that require human intervention.

Knowing the behavior of a correctly functioning system and what telemetry is generated will ensure that monitored events create alerts when the system is malfunctioning.

References

https://tools.ietf.org/html/rfc1305 Network Time Protocol (Version 3) Specification, Implementation and Analysis

https://tools.ietf.org/html/3164 The BSD syslog Protocol

https://tools.ietf.org/html/rfc3411 An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks

https://tools.ietf.org/html/rfc3412 Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)

https://tools.ietf.org/html/rfc3413 Simple Network Management Protocol (SNMP) Applications

https://tools.ietf.org/html/rfc3414 User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)

https://tools.ietf.org/html/rfc3415 View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)

https://tools.ietf.org/html/rfc3416 Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)

https://tools.ietf.org/html/rfc3417 Transport Mappings for the Simple Network Management Protocol (SNMP)

https://tools.ietf.org/html/rfc3418 Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)

Chapter 14. Troubleshooting IPsec VPNs

Introduction

When crypto is configured on Cisco IOS, at a minimum several components are used, such as IP connectivity, IKEv2, IPsec and GRE encapsulation. All these technologies are complex and often are made up of a number of components themselves. Having a base knowledge of how the protocol or feature in question operates will always help in any troubleshooting situation.

Many designers and administrators of VPN technologies soon become lost when issues occur; however, by following some simple steps they can usually find the root cause of the issue in a timely manner. If the root cause cannot be found, following a proven methodology will allow the area of issue to be determined, so it can then be examined by people with greater expertise, such as members of the Cisco Technical Assistance Center (TAC).

When the components of IPsec VPNs are segmented into the following functions and the related protocols, it supports the creation of a methodology that can be followed to correctly identify broken behavior when it is necessary to troubleshoot IPsec VPNs.

Table 14-1 illustrates the functions and protocols used when troubleshooting IPsec VPNs.

Step	Function	Protocol
1	IP connectivity	Transport routing protocol
2	VPN tunnel establishment	IKEv2
2.1	Authentication	PSK/PKI/EAP
2.2	Authorization	AAA/RADIUS
3	Data encapsulation	GRE
4	Data encryption	IPsec
5	Overlay routing	Overlay routing protocol IKEv2 routing

Table 14-1 *Troubleshooting IPsec VPN Methodology*

The functions are listed top-down, so before an IKE session can be successfully established, for example, there must be IP connectivity between both peers. This might require that static routes or a dynamic routing protocol be used. It's also worth noting that IKEv2 uses UDP port 500, with Encapsulation Security Payload and Authentication Header using IP protocol 50 and 51 respectively (or potentially UDP port 4500 if NAT Traversal is active). It could be the case that IKE traffic will be permitted; however Encapsulation Security Payload or Authentication Header could be blocked on the transport network. For this reason, IP connectivity should be tested at every stage of the protocol negotiation when the ports/protocols change.

IKEv2 (if used) will use one or more methods for authentication and authorization; these steps are listed as steps 2.1 and 2.2, which are a subset of IKEv2, because an IKE SA would not be established unless authentication and authorization were successful.

Once the IKEv2 SA is created, the subsequent IPsec Security Associations must use the same

encapsulation method. If one peer used GRE and the other a VTI, then the negotiation of the IPsec Security Association would fail. The negotiated traffic selectors and cryptographic protection must also match between peers for an IPsec Security Association to be established successfully.

Once the IPsec Security Association has been created, the relevant IP routing must be correct for traffic to be encrypted/decrypted by both parties.

Tools of Troubleshooting

There are various methods to enable troubleshooting, such as show commands, debug messages, packet capture, IKEv2 error diagnostics, and event-trace monitoring. Each has varying degrees of impact on the system when enabled. Generally, the more verbose the debugging messages are, the heavier the impact. If the output is being sent to a local buffer, it will have less impact than if it is sent to the console, which has less impact than if it is sent to a syslog server.

When troubleshooting, take care to ensure that the tools and features enabled by troubleshooting do not overload a system.

The following sections list some of the most common issues seen by Cisco TAC and the methods used to troubleshoot them. These are listed in the order of impact on the system.

Show Commands

The ability to obtain insightful information from the command-line interface (CLI) can be crucial when troubleshooting. Generally speaking running show commands to gain insight put little stress on a system.

In situations where remote access (SSH or telnet) to the CLI is obtained over a low-bandwidth link, if the output of a show command is extremely verbose, it could have an impact on the traffic over the link.

The commands listed in chapter 13, "Monitoring IPsec VPNs," can be used when troubleshooting specific aspects of the generation of an IPsec Security Association.

Syslog Messages

Messages from syslog can give an insight into normal and abnormal behavior. With a well-engineered system, normal events will generally generate a small number of syslog messages; even in failure scenarios, the telemetry gained via syslog messages can be a very lightweight method to obtain insightful information. Although the output of debugging messages is obtained via syslog, we define syslog messages as those messages obtained under normal conditions without enabling any specific debugs.

Event-Trace Monitoring

Event tracing works by reading informational messages from specific Cisco IOS software subsystems that are logged into system memory. Trace messages that are stored in memory can be displayed on the screen or saved to a file for later analysis. Messages are saved in binary format, without applying additional processing or formatting. Because of this binary format, event tracing can collect informational messages faster and for a longer time than nonbinary formats.

When a device starts behaving abnormally, it is useful to know the events that caused the abnormal

state. Turning debug on when abnormal behavior is happening generally won't help in troubleshooting, because it records data after the cause of the issue. Event-trace monitoring provides a black-box type of function that constantly records information about the state of subsystems. This feature is enabled by default and allows a historical view of events that led up to the problem state.

When debugging could potentially increase the load on an already overloaded system, event-trace monitoring allows logging information to be stored and retrieved without putting any more stress on the system.

The following example illustrates gathering information pertaining to IKEv2 and IPsec from the event-trace monitoring feature.

Under the **monitor event-trace crypto ikev2** submenu you can view errors, events, or exceptions.

Click here to view code image

```
Router#show monitor event-trace crypto ikev2 ?
  error      Show IKEv2 Errors
  event      Show IKEv2 Events
  exception  Show IKEv2 Exceptions
```

Under the **crypto ikev2 error** submenu a number of options exist. Errors can be viewed for certain time periods.

Click here to view code image

```
Router#show monitor event-trace crypto ikev2 error ?
  all         Show all the traces in current buffer
  back        Show trace from this far back in the past
  clock       Show trace from a specific clock time/date
  from-boot   Show trace from this many seconds after booting
  latest      Show latest trace events since last display
  parameters  Parameters of the trace
```

Under the **monitor event-trace crypto IPsec** submenu you can view errors, events, or exceptions

Click here to view code image

```
Router#show monitor event-trace crypto ipsec ?
  error      Show IPSEC Error
  event      Show IPSEC Events
  exception  Show IPSEC Exception
```

Similar to IKEv2, under the **crypto IPsec error** submenu a number of options exist. Errors can be viewed for certain time periods.

Click here to view code image

```
Router#show monitor event-trace crypto ipsec error ?
  all         Show all the traces in current buffer
  back        Show trace from this far back in the past
  clock       Show trace from a specific clock time/date
  from-boot   Show trace from this many seconds after booting
  latest      Show latest trace events since last display
  parameters  Parameters of the trace
```

The following example illustrates the output retrieved from the event-trace IKEv2 error conditions. This data was recorded without enabling any debugging commands.

Click here to view code image

```
Router#show monitor event-trace crypto ikev2 error all
SA ID:1 SESSION ID:1 Remote: 10.10.10.1/500 Local:
                      10.10.60.6/500
SA ID:1 SESSION ID:1 Remote: 10.10.10.1/500 Local:
                      10.10.60.6/500  : Auth exchange failed
SA ID:1 SESSION ID:1 Remote: 10.10.10.1/500 Local:
                      10.10.60.6/500  Negotiation aborted due to ERROR: Auth
                      exchange failed
ID:0 SESSION ID:0    Optional profile description not
                      updated in PSH
```

The size of the event trace or its location can be changed with the global **monitor event-trace** command. This is illustrated in the following example.

Click here to view code image

```
Router(config)#monitor event-trace crypto ikev2 error ?
  dump-file   Set name of trace dump file
  size        Set size of trace
  stacktrace  Trace call stack at tracepoints; clear the trace buffer first
  <cr>
```

Debugging

In many cases, knowing how a protocol operates gives an engineer the ability to diagnose an issue. When troubleshooting a VPN configured via IKEv2, unless the engineer is highly experienced in the configuration, in many cases debugging must be enabled to diagnose an error.

When an engineer is dealing with issues relating to an IPsec VPN using IKEv2, the most common protocols that will require debugging are IKEv2 and IPsec.

IKEv2 Debugging

Cisco IOS supports debugging of IKEv2 by allowing detailed messages to be logged for certain events pertaining to IKEv2.

IKEv2 debugging has a number of verbose options, with standard IKEv2 debugging giving information about important events when the IKEv2 Security Association is generated, such as policy, key and profile lookup, IKEv2 message exchanges, timer events, and the invocation of external components such as AAA, PKI, and callbacks.

The following example illustrates enabling standard IKEv2 debugging.

Click here to view code image

```
Router#debug crypto ikev2
IKEv2 default debugging is on
```

There are a number of more verbose options for debugging IKEv2 that are specific to certain features. These should be enabled only when issues are seen with certain features or when more verbose troubleshooting is required.

The following example illustrates the options available when enabling IKEv2 debugging.

Click here to view code image

```
Router#debug crypto ikev2 ?
  client    Client
  cluster   IKEv2 Cluster load-balancer debugging
  error     IKEv2 Error debugging
```

```
  internal  IKEv2 Internal debugging
  packet    IKEv2 Packet debugging
```

IKEv2 client debugging supports debugging of FlexVPN client features. This feature is enabled with the **debug crypto ikev2 client flexvpn** command.

IKEv2 cluster debugging supports debugging the IKEv2 cluster feature. There are two additional options, with the `detail` and `error` keywords enabling detailed verbose debugging and error conditions respectively.

IKEv2 error debugging allows error conditions to be reported. Although this debugging level is not exhaustive and will not give an exact indication of the cause of the issue, it can be used.

IKEv2 internal debugging enables the logging of debug messages for IKEv2 state machine transitions, user/group authorization attributes, merging, and config mode exchanges. This debugging level offers limited insight for users and should only be enabled by Cisco TAC.

IKEv2 packet debugging lets the IKEv2 packet be dumped to the logging facility. This lets the raw IKE data to be examined in decrypted format. The hexdump option will print the raw hex data.

The following example illustrates the output of a IKE_SA_INIT message that was obtained after enabling the **debug crypto ikev2 packet** command. After the initial message indicating the exchange type, the full IKE message is shown, with the contents of each payload described in detail.

Click here to view code image

```
IKEv2-PAK:(SESSION ID = 1,SA ID = 1):Next payload: SA, version: 2.0 Exchange type:
IKE_SA_INIT, flags: INITIATOR Message id: 0, length: 464
Payload contents:
 SA  Next payload: KE, reserved: 0x0, length: 48
  last proposal: 0x0, reserved: 0x0, length: 44
  Proposal: 1, Protocol id: IKE, SPI size: 0, #trans: 4    last transform: 0x3,
reserved: 0x0: length: 12
    type: 1, reserved: 0x0, id: AES-CBC
    last transform: 0x3, reserved: 0x0: length: 8
    type: 2, reserved: 0x0, id: SHA256
    last transform: 0x3, reserved: 0x0: length: 8
    type: 3, reserved: 0x0, id: SHA256
    last transform: 0x0, reserved: 0x0: length: 8
    type: 4, reserved: 0x0, id: DH_GROUP_2048_MODP/Group 14
 KE  Next payload: N, reserved: 0x0, length: 264
    DH group: 14, Reserved: 0x0
 N  Next payload: VID, reserved: 0x0, length: 24
 VID  Next payload: VID, reserved: 0x0, length: 23
 VID  Next payload: NOTIFY, reserved: 0x0, length: 21
 NOTIFY(NAT_DETECTION_SOURCE_IP)  Next payload: NOTIFY, reserved: 0x0, length: 28
    Security protocol id: IKE, spi size: 0, type: NAT_DETECTION_SOURCE_IP
 NOTIFY(NAT_DETECTION_DESTINATION_IP)  Next payload: NONE, reserved: 0x0, length: 28
    Security protocol id: IKE, spi size: 0, type: NAT_DETECTION_DESTINATION_IP
```

The following illustrates the output of a IKE_SA_INIT message which was obtained after enabling the **debug crypto ikev2 packet hexdump** command. After the initial message indicating the exchange type, the payload contents is described in detail. For certain payloads the data is output in raw hex. Finally we see the raw hex of the packet as sent on the wire.

Click here to view code image

```
IKEv2-PAK:(SESSION ID = 1,SA ID = 1):Next payload: SA, version: 2.0 Exchange type:
IKE_SA_INIT, flags: INITIATOR Message id: 0, length: 464
```

```
Payload contents:
 SA  Next payload: KE, reserved: 0x0, length: 48
  last proposal: 0x0, reserved: 0x0, length: 44
  Proposal: 1, Protocol id: IKE, SPI size: 0, #trans: 4    last transform: 0x3,
reserved: 0x0: length: 12
    type: 1, reserved: 0x0, id: AES-CBC
    last transform: 0x3, reserved: 0x0: length: 8
    type: 2, reserved: 0x0, id: SHA256
    last transform: 0x3, reserved: 0x0: length: 8
    type: 3, reserved: 0x0, id: SHA256
    last transform: 0x0, reserved: 0x0: length: 8
    type: 4, reserved: 0x0, id: DH_GROUP_2048_MODP/Group 14
 KE  Next payload: N, reserved: 0x0, length: 264
    DH group: 14, Reserved: 0x0

     45 B7 59 62 46 9A 5A 61 7A 93 FC 63 DC 5E 87 AC
     50 AA DB 2F 11 39 19 F5 7F E4 6C C3 57 39 60 E6
     C9 ED E4 A1 4E A1 87 87 12 79 D7 DD 68 83 06 2B
     32 25 C2 D6 01 AD F3 4F 40 23 2C 02 AA 16 6A 25
     45 B5 D2 61 B9 DD 4C 72 95 27 08 13 E4 FE E1 32
     19 F3 5A A3 0E EF 59 2A CC C4 A2 97 E4 04 73 21
     82 BB 71 4B 85 CB BE 33 55 89 61 D8 28 4A B7 CB
     32 18 AB A0 CB 5D AB C1 4D 82 3A 4B 51 41 B4 A4
     5B 56 C2 73 B3 B0 26 67 57 05 55 9D C2 BE 46 BC
     9F 1A EA 6C CA D3 07 65 02 D4 EC 32 A9 B3 94 ED
     F6 D6 EE 54 FB 74 30 F4 E3 DE E2 57 0B 50 4E 88
     E9 3C 8A 35 44 6F 34 42 DA B6 8D 49 B7 0A 8A 68
     90 A9 C7 39 C5 35 98 42 0F B4 91 F3 31 E7 42 D9
     7D 20 43 AE B5 C4 B0 34 2C 16 15 1E 9A 68 F4 F6
     C3 11 BE 46 AC C7 25 44 23 35 AF 6E EB 6B 74 CC
     14 BF F2 40 7D E6 48 B7 9A 3C C1 C4 0B A3 26 BD

  N  Next payload: VID, reserved: 0x0, length: 24

     E4 77 64 DC 98 E7 3E 85 7C 41 CA A1 21 50 2F 0F
     6C AC AE A4

  VID  Next payload: VID, reserved: 0x0, length: 23

     43 49 53 43 4F 2D 44 45 4C 45 54 45 2D 52 45 41
     53 4F 4E

  VID  Next payload: NOTIFY, reserved: 0x0, length: 21

     40 55 3D 1F 3C D1 36 43 82 0D 0F C5 15 ED 49 9E

  NOTIFY(NAT_DETECTION_SOURCE_IP)  Next payload: NOTIFY, reserved: 0x0, length: 28
    Security protocol id: IKE, spi size: 0, type: NAT_DETECTION_SOURCE_IP

     0D BE 1C 55 0D 3F A6 61 EE C8 F3 1D 8C 6B 46 9D
     E7 09 23 75

  NOTIFY(NAT_DETECTION_DESTINATION_IP)  Next payload: NONE, reserved: 0x0, length: 28
    Security protocol id: IKE, spi size: 0, type: NAT_DETECTION_DESTINATION_IP

     53 CD 98 45 F3 B9 1F 39 8E EB 2C CB C2 AA CF DE
     64 C2 85 19


IKEv2 Tx packet(on wire):
```

```
F4 9D AD A2 60 CB 37 3F 00 00 00 00 00 00 00 00
21 20 22 08 00 00 00 00 00 00 01 D0 22 00 00 30
00 00 00 2C 01 01 00 04 03 00 00 0C 01 00 00 0C
80 0E 00 80 03 00 00 08 02 00 00 05 03 00 00 08
03 00 00 0C 00 00 00 08 04 00 00 0E 28 00 01 08
00 0E 00 00 45 B7 59 62 46 9A 5A 61 7A 93 FC 63
DC 5E 87 AC 50 AA DB 2F 11 39 19 F5 7F E4 6C C3
57 39 60 E6 C9 ED E4 A1 4E A1 87 87 12 79 D7 DD
68 83 06 2B 32 25 C2 D6 01 AD F3 4F 40 23 2C 02
AA 16 6A 25 45 B5 D2 61 B9 DD 4C 72 95 27 08 13
E4 FE E1 32 19 F3 5A A3 0E EF 59 2A CC C4 A2 97
E4 04 73 21 82 BB 71 4B 85 CB BE 33 55 89 61 D8
28 4A B7 CB 32 18 AB A0 CB 5D AB C1 4D 82 3A 4B
51 41 B4 A4 5B 56 C2 73 B3 B0 26 67 57 05 55 9D
C2 BE 46 BC 9F 1A EA 6C CA D3 07 65 02 D4 EC 32
A9 B3 94 ED F6 D6 EE 54 FB 74 30 F4 E3 DE E2 57
0B 50 4E 88 E9 3C 8A 35 44 6F 34 42 DA B6 8D 49
B7 0A 8A 68 90 A9 C7 39 C5 35 98 42 0F B4 91 F3
31 E7 42 D9 7D 20 43 AE B5 C4 B0 34 2C 16 15 1E
9A 68 F4 F6 C3 11 BE 46 AC C7 25 44 23 35 AF 6E
EB 6B 74 CC 14 BF F2 40 7D E6 48 B7 9A 3C C1 C4
0B A3 26 BD 2B 00 00 18 E4 77 64 DC 98 E7 3E 85
7C 41 CA A1 21 50 2F 0F 6C AC AE A4 2B 00 00 17
43 49 53 43 4F 2D 44 45 4C 45 54 45 2D 52 45 41
53 4F 4E 29 00 00 15 46 4C 45 58 56 50 4E 2D 53
55 50 50 4F 52 54 45 44 29 00 00 1C 01 00 40 04
0D BE 1C 55 0D 3F A6 61 EE C8 F3 1D 8C 6B 46 9D
E7 09 23 75 00 00 00 1C 01 00 40 05 53 CD 98 45
F3 B9 1F 39 8E EB 2C CB C2 AA CF DE 64 C2 85 19
```

IPsec Debugging

Although IKEv2 provides insight into the traffic selectors that are exchanged to create an IPsec Security Association, detailed debugging of IPsec can be enabled separately.

IPsec debugging will give details of the protocol (Encapsulation Security Payload or Authentication Header), transform, and SPI used when constructing the IPsec Security Association.

IPsec debugging can be enabled with the **debug crypto ipsec** command.

A number of more verbose IPsec debugging commands can be enabled. The following example illustrates the additional options available when enabling IPsec debugging.

Click here to view code image

```
Router#debug crypto ipsec ?
  client      Client Debug
  error       IPSEC errors
  ha          IPSEC High Availability
  hw-request  IPSEC hw-request
  message     IPSEC message
  metadata    CTS metadata
  states      IPSEC states
```

Key Management Interface Debugging

Key Management Interface (KMI) is a message-based programmatic interface between the IOS IPsec and key management processes such as IKEv1 and IKEv2. KMI defines a set of well-defined messages between IKE and IPsec.

Table 14-2 describes the KMI messages sent from IPsec to IKEv2. Note that the KMI messages use the KEY_ENG prefix.

KMI Message from IPsec to IKEv2	Description
KEY_ENG_REQUEST_SAS	This message is used on an initiator by IPsec to request the initiation of a new IPsec Security Association. The message is also used by IPsec to initiate the rekey of an existing IPsec Security Association on both initiator and responder.
KEY_ENG_DELETE_SAS	This message is used by IPsec to notify IKEv2 of deletion of an IPsec Security Association. The deletion may be triggered by user command, SA expiry or initiated by the peer
KEY_ENG_IPSEC_READY	This message is used on a dVTI responder by IPsec to notify IKEv2 of successful creation of a virtual-access interface with tunnel protection applied on it. This message is sent in response to KEY_MGR_IKMP_READY from IKEv2 that requests IPsec for virtual-access interface creation for the IKE/IPsec session being negotiated.
KEY_ENG_NOTIFY_INCR_COUNT	This message is used by IPsec to notify IKEv2 of successful installation of an IPsec Security Association

Table 14-2 *KMI Messages Sent from IPsec to IKEv2*

Table 14-3 describes the KMI messages sent from IKEv2 to IPsec. Note that the KMI messages use the KEY_MGR prefix.

KMI Message from IKEv2 to IPsec	Description
KEY_MGR_IKMP_READY	This message is used on a dVTI responder by IKEv2 to request IPsec for virtual-access interface creation for the IKE/IPsec session being negotiated.
KEY_MGR_VALIDATE_IPSEC_PROPOSALS	This message is used by IKEv2 to request IPsec to validate the IPsec Security Association proposal from the peer and find an IPsec interface that can accept the proposal.
KEY_MGR_CREATE_IPSEC_SAS	This message is used by IKEv2 to request IPsec to install the IPsec Security Association that was successfully negotiated.
KEY_MGR_DELETE_SAS	This message is used by IKEv2 to request IPsec to delete an IPsec Security Association on receipt of a SA delete message from the peer
KEY_MGR_SESSION_CLOSED	This message is used by IKEv2 to notify IPsec of the deletion of the IKE/IPsec session and IPsec would delete the associated virtual-access interface

Table 14-3 *KMI Messages Sent from IKEv2 to IPsec*

KMI debugging is a very useful debugging tool in isolating a problem between IKE and IPsec. The KMI debugging is enabled using the **debug crypto kmi** command.

The following example illustrates the **debug crypto kmi** output for an IKE/IPsec session establishment on an IPsec sVTI-based initiator.

Click here to view code image

```
KMI: (Session ID: 1) IPSEC key engine sending message KEY_ENG_REQUEST_SAS to
Crypto IKEv2.
KMI: (Session ID: 1) Crypto IKEv2 received message KEY_ENG_REQUEST_SAS from IPSEC
key engine.
KMI: (Session ID: 1) Crypto IKEv2 sending message KEY_MGR_VALIDATE_IPSEC_PROPOSALS
to IPSEC key engine.
KMI: (Session ID: 1) IPSEC key engine received message KEY_MGR_VALIDATE_IPSEC_
PROPOSALS from Crypto IKEv2.
KMI: (Session ID: 1) Crypto IKEv2 sending message KEY_MGR_CREATE_IPSEC_SAS to
IPSEC key engine.
KMI: (Session ID: 1) IPSEC key engine received message KEY_MGR_CREATE_IPSEC_SAS
from Crypto IKEv2.
KMI: IPSEC key engine sending message KEY_ENG_NOTIFY_INCR_COUNT to Crypto IKEv2.
```

The following example illustrates the **debug crypto kmi** output for an IKE/IPsec session establishment on an IPsec dVTI-based responder.

Click here to view code image

```
KMI: (Session ID: 3) Crypto IKEv2 sending message KEY_MGR_IKMP_READY to IPSEC key
engine.
```

```
KMI: (Session ID: 3) IPSEC key engine received message KEY_MGR_IKMP_READY from
Crypto IKEv2.

KMI: IPSEC key engine sending message KEY_ENG_IPSEC_READY to Crypto IKEv2.
KMI: Crypto IKEv2 received message KEY_ENG_IPSEC_READY from IPSEC key engine.
KMI: (Session ID: 3) Crypto IKEv2 sending message KEY_MGR_VALIDATE_IPSEC_PROPOSALS
to IPSEC key engine.
KMI: (Session ID: 3) IPSEC key engine received message
KEY_MGR_VALIDATE_IPSEC_PROPOSALS from Crypto IKEv2.
KMI: (Session ID: 3) Crypto IKEv2 sending message
KEY_MGR_CREATE_IPSEC_SAS to IPSEC key engine.
KMI: (Session ID: 3) IPSEC key engine received message KEY_MGR_CREATE_IPSEC_SAS
from Crypto IKEv2.
KMI: IPSEC key engine sending message KEY_ENG_NOTIFY_INCR_COUNT to Crypto IKEv2.
KMI: Crypto IKEv2 received message KEY_ENG_NOTIFY_INCR_COUNT from IPSEC
key engine.
```

PKI Debugging

Public key infrastructure (PKI) debugging events can be generated to give detailed information about certificate enrollment or renewal. The applications that use PKI, such as IKEv2, can generate specific debug messages when using the PKI trustpoint to perform signing and/or validation. Debugging can be enabled to give detailed information regarding the PKI's interaction with the crypto engine when performing operations with the cryptographic key material.

Conditional Debugging

By default Cisco IOS provides a debugging mechanism that works in an all-or-none model, whereby specific debugging is either enabled or disabled. This is not efficient in debugging problems in a large-scale VPN deployment, where many remote devices are connecting to a hub, causing a large number of debugging messages to be generated.

Conditional debugging allows a user to define a set of condition filters, so that the system outputs only debug messages related to those predefined conditions based on filter checking at run time. The debug filters include a number of parameters, such as peer IP address, destination (subnet with mask), connection-id, and SPI.

When multiple debug conditions are set, the debug message is printed only if all the conditions are met.

When a debug message would be output, if the debug condition parameter is not available for comparison, then the debug message is suppressed. For example, when conditional debugging is based on an IKEv2 profile, the IKEv2 profile is available on a responder only after it receives the IKE_AUTH request (in the third message), so until then the messages are suppressed. Be aware that in this instance you will not see any debug message that occurred in the IKE_SA_INIT exchange.

This behavior can be changed with the **debug crypto condition unmatched** command. In this instance all available messages are generated, but this can soon fill most log buffers for busy systems.

Click here to view code image

```
Router#debug crypto condition ?
  unmatched  Output debugs even if no context available
```

Conditional debugging can be used to minimize the risk of consuming CPU cycles and to create debug messages that pertain to certain peers only.

The **debug crypto condition ikev2 profile *profile_name*** command can be used to enable IKEv2 debugging for peers that associate with a certain IKEv2 profile.

The **debug crypto condition peer [group** *group***] [hostname** *hostname***] [ipv4** *ipaddress***] [subnet** *subnet mask***] [username** *username***]]** command can be used to enable debug messages to be generated for one or more remote peers. This is a very handy command when troubleshooting connectivity issues on VPN gateways that are serving multiple clients. When troubleshooting issues on VPN gateways with many active clients, if conditional debugging is not used the amount of logging information generated can overwhelm many administrators.

Conditional debugging can be seen as a filter and should be used in conjunction with IKEv2 or IPsec debugging.

IP Connectivity

IKEv2 uses UDP as a transport protocol over port 500 or port 4500 (when NAT is detected). When NAT or PAT is used on the transport network, the source port used by the UDP traffic can be translated from UDP/500 to any UDP port. When IPsec VPNs are deployed over the Internet or a network that uses NAT or PAT, access control lists that pertain to IKEv2 should be configured to allow UDP traffic from any source port.

To validate IP connectivity between hosts, the ping command, which uses ICMP echo and echo-reply, can be used to check that an IP-level path exists. It should be noted that if a firewall or access-control device permits ICMP but not the protocols and/or ports used by IKE, Encapsulation Security Payload, or Authentication Header, then the results of the ping command might be misleading.

We must assume that the physical interface that is used as the source of the tunnel is physically connected and in the up state; this might seem an obvious check, but if the interface is down for any reason, then traffic cannot be sent or received on the physical network interface card (NIC).

There are cases where traffic is fragmented at the IP layer with fragments being lost in transit. In this case the fragmented traffic is never reassembled, so it will not be passed up the stack and hence appear in IKE debugs. Embedded Packet Capture (EPC) is an onboard packet capture facility that enables the system to capture packets flowing into, through, and from the device. The captured traffic can be analyzed locally or saved and exported for offline analysis by using a tool such as Wireshark.

The following example illustrates configuring an EPC to capture traffic on an interface.

A capture buffer is defined, which is a temporary buffer for storing the captured packets. Various options can be selected when the buffer is defined, such as size, maximum packet size, and circular/linear. In this example, the maximum size of packets is set to 1600 bytes to ensure the full packet is captured. Depending on the size of MTU and the requirement as to what is to be captured, this value should be adjusted accordingly.

Click here to view code image

```
Router#monitor capture buffer buf1 size 2048 max-size 1600 linear
```

Note that if certain traffic is being captured, a filter can also be applied to limit the capture to desired traffic. Define an access control list (ACL) within config mode and apply the filter to the buffer:

Click here to view code image

```
Router(config)#ip access-list buf-filter
```

```
Router(config-ext-nacl)#permit udp any any eq 500

Router#monitor capture buffer buf1 filter access-list buf-filter
```

A capture point is created, which defines the location where the capture occurs. In this case only incoming traffic is captured. The capture point also defines whether the capture occurs for IPv4 or IPv6 and in which switching path.

Click here to view code image

```
Router#monitor capture point ip cef point1 GigabitEthernet0/0 in
```

The buffer is attached to the capture point:

Click here to view code image

```
Router#monitor capture point associate point1 buf1
```

The capture is started:

Click here to view code image

```
Router#monitor capture point start point1
%BUFCAP-6-ENABLE: Capture Point point1 enabled
```

Once the capture is active, ensure that the traffic to be captured is generated. This could be as simple as disabling and enabling a tunnel interface (using shutdown and no shutdown commands) to ensure that IKE traffic is generated. Once the intended traffic has been generated, the capture can be stopped.

Click here to view code image

```
Router#monitor capture point stop point1
%BUFCAP-6-DISABLE: Capture Point point1 disabled
```

The contents of the capture can be viewed (this data is in hexadecimal). This hex information can be converted into a human-readable packet format (for example, pcap) using any tool that converts raw hexadecimal into pcap format.

Click here to view code image

```
Router#show monitor capture buffer buf1 dump
UTC May 21 2016 : IPv4 LES CEF    : Gi0/0 None

2D2EBFA0:                           A44C110E              $L..
2D2EBFB0: B4A0C89C 1DFEBF20 080045C0 00420058  4 H..~? ..E@.B.X
2D2EBFC0: 0000FE11 610DAC10 0101AC10 016401F4  ..~.a.,...,..d.t
<removed>
```

The capture can be exported in pcap format so it can then be opened in a tool such as Wireshark.

Click here to view code image

```
monitor cap buffer buf1 export flash:/buf1.pcap
```

The following example shows the IKEv2 debugging output from a client that does not have IP connectivity with its peer. We see that the IKE_SA_INIT packet is sent; however, this packet is retransmitted a number of times, which is a sure indication that the packet did not reach the peer, or if it did, the returning packet was not received.

Click here to view code image

```
IKEv2:(SESSION ID = 1,SA ID = 1):IKE Proposal: 1, SPI size: 0 (initial
negotiation),
Num. transforms: 15
   AES-CBC   AES-CBC   AES-CBC   SHA512
SHA384   SHA256   SHA1   MD5   SHA512   SHA384   SHA256   SHA96   MD596   DH_GROUP_153
5
DH_GROUP_1024_MODP/Group 2

 IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 10.10.10.1:500/From
10.10.20.3:500/VRF i0:f0]
Initiator SPI : 502A6C10FFD6F8BC - Responder SPI : 0000000000000000 Message id: 0

IKEv2 IKE_SA_INIT Exchange REQUEST
Payload contents:
 SA KE N VID VID NOTIFY(NAT_DETECTION_SOURCE_IP)
NOTIFY(NAT_DETECTION_DESTINATION_IP)

IKEv2:(SESSION ID = 1,SA ID = 1):Insert SA
FlexClient#
IKEv2:(SESSION ID = 1,SA ID = 1):Retransmitting packet

IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 10.10.10.1:500/From
10.10.20.3:500/VRF i0:f0]
Initiator SPI : 502A6C10FFD6F8BC - Responder SPI : 0000000000000000 Message id: 0
IKEv2 IKE_SA_INIT Exchange REQUEST
Payload contents:
 SA KE N VID VID NOTIFY(NAT_DETECTION_SOURCE_IP)
NOTIFY(NAT_DETECTION_DESTINATION_IP)

IKEv2:(SESSION ID = 1,SA ID = 1):Retransmitting packet
```

In this case, the reason why IP connectivity failed between both peers must be determined.

There are cases where IP connectivity can be lost within the IKE_AUTH exchange. When certificates are used, if the IKE datagram becomes fragmented and IKEv2 fragmentation is not used, then connectivity can fail. In the following example we see that the initiator sends the IKE_AUTH packet; however, the reply to this is not received, and the IKE_AUTH is retransmitted a number of times until it fails. This can be an indication that fragmentation occurred, for the IKE_SA_INIT exchange was successful.

Click here to view code image

```
IKEv2:(SESSION ID = 1,SA ID = 1):Building packet for encryption.
Payload contents:
 VID IDi CERT CERTREQ NOTIFY(HTTP_CERT_LOOKUP_SUPPORTED) AUTH CFG SA TSi TSr
NOTIFY(INITIAL_CONTACT) NOTIFY(USE_TRANSPORT_MODE) NOTIFY(SET_WINDOW_SIZE)
NOTIFY(ESP_TFC_NO_SUPPORT) NOTIFY(NON_FIRST_FRAGS)

IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 10.10.10.1:500/From
10.10.50.5:500/VRF i0:f0]
Initiator SPI : DB0340BE40E55EE9 - Responder SPI : 756425ADB92B135A Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST
Payload contents:
 ENCR

IKEv2:(SESSION ID = 1,SA ID = 1):Retransmitting packet

IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 10.10.10.1:500/From
10.10.50.5:500/VRF i0:f0]
Initiator SPI : DB0340BE40E55EE9 - Responder SPI : 756425ADB92B135A Message id: 1
```

```
IKEv2 IKE_AUTH Exchange REQUEST
Payload contents:
 ENCR

%LINK-3-UPDOWN: Interface Tunnel1, changed state to up
IKEv2:(SESSION ID = 1,SA ID = 1):Retransmitting packet

IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 10.10.10.1:500/From
10.10.50.5:500/VRF i0:f0]
Initiator SPI : DB0340BE40E55EE9 - Responder SPI : 756425ADB92B135A Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST
Payload contents:
 ENCR
```

Note

If a VRF-aware IPsec is used in which the IVRF and FVRF are different, then ensure that the peer's tunnel source address is reachable via the FVRF. This is a common mistake when VRF-aware IPsec is used. The **show ip route vrf** *vrf_name* command can be used to examine the FVRF routing table.

VPN Tunnel Establishment

IKEv2 is the protocol used to negotiate cryptographic algorithms in the IKE_SA_INIT exchange, authenticate both parties and establish IPsec Security Associations in the IKE_AUTH exchange, maintain the IKEv2 SA using INFORMATIONAL messages, and rekey using the CREATE_CHILD_SA exchange. Based on this knowledge, the steps involved in the creation of an IKEv2 SA can be followed, and when the observed behavior is not as intended debugging can be enabled to diagnose the cause of the failure.

IKEv2 Diagnose Error

If there is an error in the IKEv2 exchange, this will be logged internally. The errors can be viewed using the **show crypto ikev2 diagnose error** command. The following example illustrates a number of error conditions which were generated on an IKEv2 node.

Click here to view code image

```
Router#show crypto ikev2 diagnose error
Exit Path Table - status: enable, current entry 4, deleted 0, max allow 50

Error(1): Failed to locate an item in the database

-Traceback= 9D64A28z 8CBE65Ez 8C66C40z 8CA6FF5z 8C746E0z 8C5359Az 8C5A25Ez
8C59F5Fz 8C97FE2z 8C979DBz 8C6A95Dz 8CC2BC3z

Error(1): A supplied parameter is incorrect

-Traceback= 9D64A28z 8CBE65Ez 8C66C40z 8CC273Ez 8CFA2F1z 8CF4038z 8C8B7F2z
8C7B455z 8C7AFE0z 8C5549Dz 8C5A25Ez 8C59F5Fz 8C6FDCAz 8C6EC98z 8C6A95Dz 8CC2BC3z

Error(1): AAA author not configured in IKEv2 profile

-Traceback= 9D64A28z 8CBE65Ez 8C66C40z 8CC6D2Dz 8CC6AC2z 8CC9661z 8CC9537z
8CC2B8Fz
```

```
Error(1): We requested a certificate, but the peer supplied none

-Traceback= 9D64A28z 8CBE65Ez 8C66C40z 8C7BABBz 8C7B6B8z 8C553F3z 8C5A25Ez
8C59F5Fz 8C97FE2z 8C979DBz 8C6A95Dz 8CC2BC3z
```

The diagnose error message log can be cleared with the **clear crypto ikev2 diagnose error** command. This allows for errors to be cleared, connectivity tested, and then errors checked again.

As the same error message can be generated from multiple places in the code, the traceback decode helps Cisco TAC determine the exact location where the message was generated.

Troubleshooting the IKE_SA_INIT Exchange

The following example looks to highlight how misconfigured proposals can cause the IKE_SA_INIT exchange to fail. We will focus on two devices that are misconfigured.

As both parties are negotiating the cryptographic algorithms that will be used to secure the IKEv2 SA, a common issue that causes the IKE_SA_INIT exchange not to be successful is mismatching policies between peers.

Table 14-4 details the proposal that is configured on the initiator.

Type	Algorithm
Encryption	AES-CBC-128
Pseudo Random Function	SHA512
Integrity	SHA512
DH Group	DH Group 1

Table 14-4 *Initiator Proposal*

Table 14-5 details the proposal that is configured on the responder.

Type	Algorithm
Encryption	AES-CBC-128
Pseudo Random Function	SHA256
Integrity	SHA256
DH Group	DH Group 14

Table 14-5 *Responder Proposal*

Obviously these attributes do not match, so an IKEv2 SA will not be established.

On the initiator, we see that the IKE_SA_INIT exchange is started, and the proposal sent to the peer,

Click here to view code image

```
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Start PKI Session
IKEv2:(SA ID = 1):[PKI -> IKEv2] Starting of PKI Session PASSED
IKEv2:(SESSION ID = 1,SA ID = 1):[IKEv2 -> Crypto Engine] Computing DH public key,
DH Group 1
IKEv2:(SA ID = 1):[Crypto Engine -> IKEv2] DH key Computation PASSED
IKEv2:(SESSION ID = 1,SA ID = 1):Request queued for computation of DH key
IKEv2:IKEv2 initiator - no config data to send in IKE_SA_INIT exch
IKEv2:(SESSION ID = 1,SA ID = 1):Generating IKE_SA_INIT message
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):IKE Proposal: 1, SPI size: 0
(initial negotiation),
Num. transforms: 4
   AES-CBC   SHA512   SHA512   DH_GROUP_768_MODP/Group 1

IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 10.10.10.1:500/From
10.10.50.5:500/VRF i0:f0]
Initiator SPI : D007314789FFB97A - Responder SPI : 0000000000000000 Message id: 0
IKEv2 IKE_SA_INIT Exchange REQUEST
Payload contents:
 SA KE N VID VID NOTIFY(NAT_DETECTION_SOURCE_IP)
NOTIFY(NAT_DETECTION_DESTINATION_IP)

IKEv2:(SESSION ID = 1,SA ID = 1):Insert SA
```

On the responder when the proposal is not compatible, we see that the IKE_SA_INIT message was received. However the policies within this did not match the locally configured proposal, so the NO_PROPOSAL_CHOSEN message is sent.

Click here to view code image

```
IKEv2:(SESSION ID = 9,SA ID = 1):Processing IKE_SA_INIT message
IKEv2:(SESSION ID = 9,SA ID = 1):Received Policies: : Failed to find a matching
policyProposal 1:  AES-CBC-128 SHA512 SHA512 DH_GROUP_768_MODP/Group 1

IKEv2:(SESSION ID = 9,SA ID = 1):Expected Policies: : Failed to find a matching
policyProposal 1:  AES-CBC-128 SHA256 SHA256 DH_GROUP_2048_MODP/Group 14

Proposal 2:  AES-CBC-128 SHA256 SHA256 DH_GROUP_2048_MODP/Group 14

IKEv2:(SESSION ID = 9,SA ID = 1):: Failed to find a matching policy
IKEv2:(SESSION ID = 9,SA ID = 1):Sending no proposal chosen notify
```

The responder generates a reply containing a Notify payload of type NO_PROPOSAL_CHOSEN, which indicates that the proposals were not compatible.

The following example illustrates the debug messages seen on the initiator, where the notify payload sent from the responder can be seen, indicating that no proposal was chosen.

Click here to view code image

```
IKEv2:(SESSION ID = 1,SA ID = 1):Received Packet [From 10.10.10.1:500/To
10.10.50.5:500/VRF i0:f0]
Initiator SPI : D007314789FFB97A - Responder SPI : 63922088658CEA7B Message id: 0

IKEv2 IKE_SA_INIT Exchange RESPONSE
Payload contents:
 NOTIFY(NO_PROPOSAL_CHOSEN)
```

To rectify this situation, ensure that the IKEv2 proposals are compatible between both peers. The **show crypto ikev2 proposal** command can be used to verify the configured IKEv2 proposals. The IKEv2 proposal can be changed by configuring the relevant cryptographic algorithms within the IKEv2 proposal. The following example illustrates that the incorrect attributes are configured:

Click here to view code image

```
Router#show crypto ikev2 proposal
 IKEv2 proposal: default
     Encryption : AES-CBC-128
     Integrity  : SHA512
```

```
     PRF        : SHA512
     DH Group   : DH_GROUP_768_MODP/Group 1
```

The following example illustrates that the relevant IKEv2 proposal is then changed to include the correct integrity and Diffie-Hellman group.

Click here to view code image

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router (config)#crypto ikev2 proposal default
Router (config-ikev2-proposal)#integrity sha256
Router (config-ikev2-proposal)#group 14
Router #show crypto ikev2 proposal
 IKEv2 proposal: default
     Encryption : AES-CBC-128
     Integrity  : SHA256
     PRF        : SHA256
     DH Group   : DH_GROUP_2048_MODP/Group 14
```

Note

Within the IKE_SA_INIT exchange, cookie notification can be used. If a responder replied with the cookie notification payload and the initiator did not understand this for some reason, then connectivity would fail.

Note

The IKEv2 redirect feature can be used in the IKE_SA_INIT exchange; having a client redirected to another VPN gateway should occur only if supported by both the initiator and the responder.

Troubleshooting the IKE_AUTH Exchange

Within the IKE_AUTH exchange, the peers authenticate each other. Each peer also exchanges traffic selectors that are negotiated to form the IPsec Security Association.

Within IKEv2, the authentication is bidirectional and not negotiated. Both the initiator and responder must be configured as to what authentication method is used. This requires that the method the initiator uses to authenticate itself and the method that is set on the responder must match.

The following example illustrates the syslog messages that are generated on a responder when debugging an IKEv2 negotiation where the authentication method proposed by the initiator does not match what is configured on the responder.

Click here to view code image

```
IKEv2:(SESSION ID = 32,SA ID = 2):Get peer's authentication method
IKEv2:(SESSION ID = 32,SA ID = 2):: Peer's configured auth method mis-matches with
proposed auth method
IKEv2:(SESSION ID = 32,SA ID = 2):Verification of peer's authentication data
FAILED
IKEv2:(SESSION ID = 32,SA ID = 2):Sending authentication failure notify
IKEv2:(SESSION ID = 32,SA ID = 2):Building packet for encryption.
Payload contents:
 NOTIFY(AUTHENTICATION_FAILED)
```

To rectify this situation, the correct authentication method needs to be configured on both the initiator and the responder. This can be achieved by the **authentication** command within the IKEv2 profile.

Authentication

There are three methods of authentication when IKEv2 is used: digital signature using RSA or ECDSA, PSK, and EAP.

Troubleshooting RSA or ECDSA Authentication

As authentication with certificates involves a number of components, we shall recap how this form of authentication occurs. Having an understanding of the traffic flows provides a step-by-step method to follow when troubleshooting.

Figure 14-1 illustrates the decisions that occur when digital signatures are used as the authentication method.

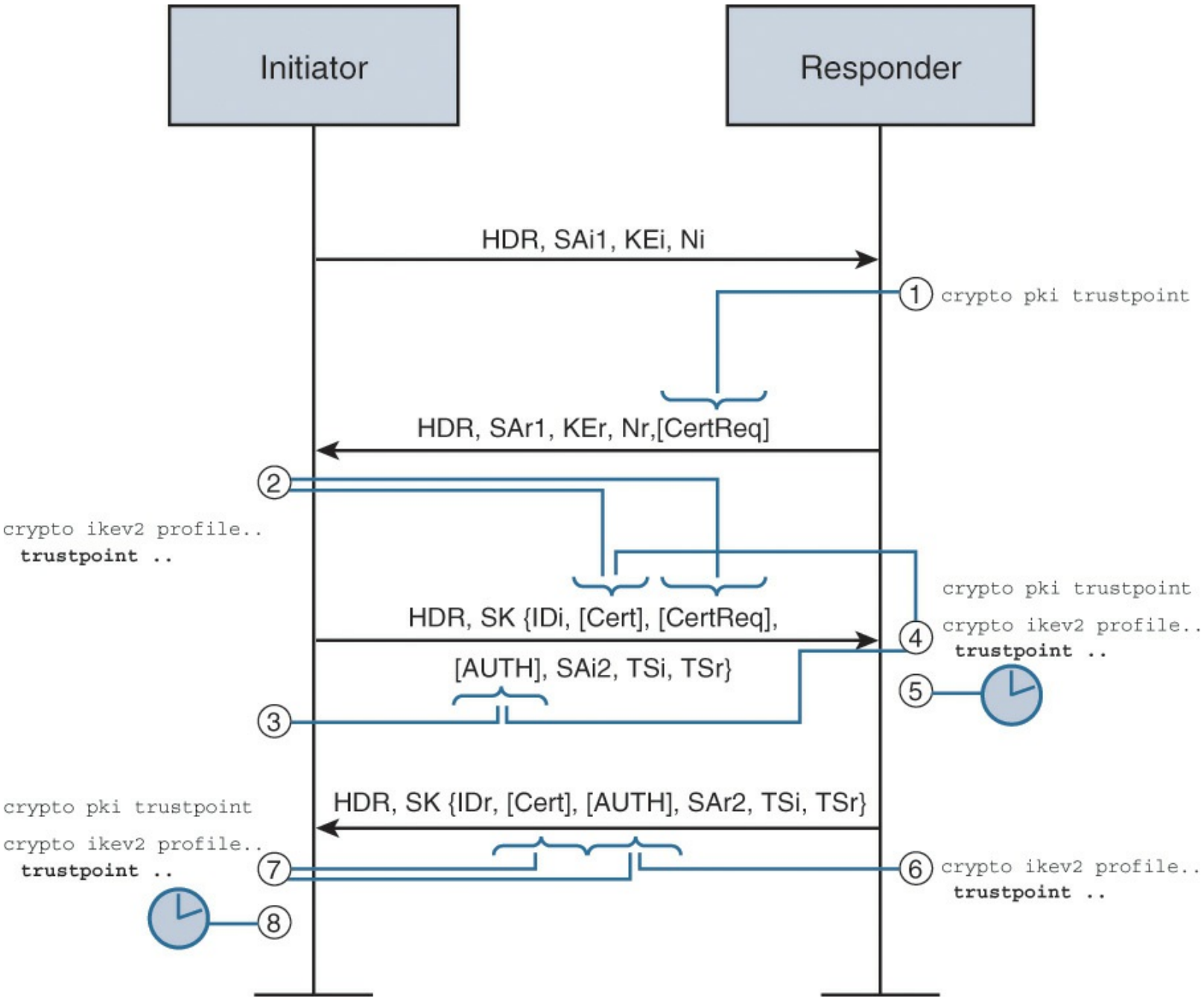


Figure 14-1 *RSA/ECDSA authentication*

1. Assuming that both peers are using RSA or EC digital signatures to prove their authenticity, within the IKE_SA_INIT exchange the responder gives a hint to the initiator about the trustpoints it supports. This is achieved by sending a SHA-1 hash of the public key of the certificate authority contained within the CERTREQ payload.
2. When the initiator begins the IKE_AUTH exchange, if the responder indicated a PKI that is configured within the IKEv2 profile, this PKI is selected; otherwise the first trustpoint configured within the IKEv2 profile is used.
3. The initiator uses the private key that relates to the certificate of the selected trustpoint that is used to sign the AUTH payload. The AUTH payload is sent to the peer, along with the associated certificate, in the CERT payload. The CERTREQ payload is also included by the initiator in the IKE_AUTH exchange; this allows a peer with multiple trustpoints configured to select the correct one.
4. The responder will check that it has a valid trustpoint configured in the IKEv2 profile and for the certificate presented. The responder will check the digital signature of the presented certificate to ensure that the certificate authority signed this.
5. The certificate's not-before and not-after dates are checked to ensure that the certificate is being used within the permitted time. If revocation checking is enabled, then a Certificate Revocation List (CRL) will be queried, or an Online Certificate Status Protocol (OCSP) request will be sent. The AUTH payload is verified using the public key contained within the certificate to validate that it was signed using the corresponding private key.
6. Assuming that the authentication was successful, the responder will itself prove its authenticity by signing the AUTH payload; the trustpoint used will be selected from what the initiator indicated it supports within the CERTREQ payload. The AUTH payload is then sent in the IKE_AUTH response message.
7. On receipt of this payload, the initiator will check that it has a valid trustpoint configured in the IKEv2 profile and for the certificate presented. The initiator will check the digital signature of the presented certificate to ensure that the certificate authority signed this.
8. The certificate's not-before and not-after dates are checked to ensure that the certificate is being used within the permitted time. If a revocation checking is enabled then a Certificate Revocation List (CRL) will be queried, or an Online Certificate Status Protocol (OCSP) request is sent. The AUTH payload is verified using the public key contained within the certificate to validate that it was signed using the corresponding private key.

With Cisco IOS, the IKE subsystem will make calls to the PKI subsystem to perform the previously described cryptographic processes. Once this process has been completed, the PKI subsystem will signal to IKE if it was successful or not. We will now examine potential issues that can occur within the previously described steps.

When the responder receives an IKE_SA_INIT, it will reply with the CERTREQ to inform the peer of the supported trustpoints. If the initiator is not enrolled in any of these, then the message in the following example will be seen on the initiator if IKEv2 debugging (**debug crypto ikev2**) is enabled. The responder will then use the certificate and related key material of first configured trustpoint within the IKEv2 profile to sign the authentication payload.

Click here to view code image

```
IKEv2:(SESSION ID = 1,SA ID = 1):Processing IKE_SA_INIT message
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Retrieving trustpoint(s) from received
certificate hash(es)
IKEv2:(SA ID = 1):[PKI -> IKEv2] Retrieved trustpoint(s): NONE
IKEv2:% Received cert hash is invalid, using configured trustpoints from profile
for signing
```

The configured trustpoints can be shown with the **show crypto pki trustpoints [status** | *label* **[status]]** command.

The following example illustrates a correctly configured trustpoint, with corresponding RSA keypair.

Click here to view code image

```
FlexClient#show crypto pki trustpoints PKI status
Trustpoint PKI:
  Issuing CA certificate configured:
    Subject Name:
     cn=ca.cisco.com
    Fingerprint MD5: 8E628B19 7251C76A 543A76B0 8961EAEB
    Fingerprint SHA1: 46D32784 A5F7CF3F 2CC86337 6B0221E5 A5C5B4CD
  Router General Purpose certificate configured:
    Subject Name:
     hostname=flexclient.cisco.com
    Fingerprint MD5: DA640991 D2BA5328 F05594AE 09E2E235
    Fingerprint SHA1: 67C92EAB 24C7CDA1 B28E2696 1C5593D2 A23F0AB2
  State:
    Keys generated ..... Yes (General Purpose, non-exportable)
    Issuing CA authenticated ..... Yes
    Certificate request(s) ..... Yes
```

To view the local certificates, use the **show crypto pki certificates [** *trustpoint-name* **[verbose]]** command.

The following example illustrates viewing two certificates that are associated with a trustpoint. The local certificate that is used to authenticate the device along with the CA certificate is shown. This is a very handy command for verifying that the certificate has not expired.

Click here to view code image

```
FlexClient#show crypto pki certificates PKI
Certificate
  Status: Available
  Certificate Serial Number (hex): 02
  Certificate Usage: General Purpose
  Issuer:
    cn=ca.cisco.com
  Subject:
    Name: flexclient.cisco.com
    hostname=flexclient.cisco.com
  Validity Date:
    start date: 19:03:45 JST Dec 21 2015
    end   date: 19:03:45 JST Dec 21 2016
  Associated Trustpoints: PKI
  Storage: nvram:s1#PKI.cer

CA Certificate
  Status: Available
  Certificate Serial Number (hex): 01
  Certificate Usage: Signature
  Issuer:
```

```
    cn=ca.cisco.com
  Subject:
    cn=ca.cisco.com
  Validity Date:
    start date: 19:02:16 JST Dec 21 2015
    end   date: 19:02:16 JST Dec 21 2018
  Associated Trustpoints: 20 PKI
  Storage: nvram:s1#1CA.cer
```

If the verbose keyword is used, more detailed output can be viewed. This command will show the key usage, which indicates for what purpose the key within the certificate can be used. The following example illustrates viewing the key usage fields of a certificate.

Click here to view code image

```
FlexClient#show crypto pki certificates verbose
Certificate
  Status: Available
  Version: 3
  Certificate Serial Number (hex): 02
  ...
  X509v3 extensions:
    X509v3 Key Usage: A0000000
      Digital Signature
      Key Encipherment
    X509v3 Subject Key ID: 193A5E31 B1B5DECB C1172311 A61FEF7E E049CED0
```

When the Cisco AnyConnect client is used, name verification is performed on the server certificate. The following rules are applied for the purposes of IPsec name verification:

- If a Subject Alternative Name extension is present with relevant attributes, name verification is performed solely against the Subject Alternative Name. Relevant attributes include DNS Name attributes for all certificates, and additionally include IP address attributes if the connection is being performed to an IP address.
- If a Subject Alternative Name extension is not present or is present but contains no relevant attributes, name verification is performed against any common name attributes found in the subject of the certificate.
- If a certificate uses a wildcard for the purposes of name verification, the wildcard must be in the first (left-most) subdomain only, and additionally it must be the last (right-most) character in the subdomain. Any wildcard entry not in compliance is ignored for the purposes of name verification.

Certificate Attributes

Key Usage is an X509v3 certificate extension that is described in RFC2459, “Internet X.509 Public Key Infrastructure Certificate and CRL Profile.” The key usage extension defines the purpose (for example, encipherment, signature, certificate signing) of the key contained in the certificate.

RFC4945, “The Internet IP Security PKI Profile of IKEv1/ISAKMP, IKEv2, and PKIX,” describes the Key Usage attribute when using IKEv2.

According to the following extract from RFC4945, it is standard practice to set the key usage to digital signature; however, Cisco IOS will also set key encipherment. It is recommended to set both attributes.

Click here to view code image

```
Since we are talking about using the public key to validate a signature, if the
KeyUsage extension is present, then at least one of the digitalSignature or the
nonRepudiation bits in the KeyUsage extension MUST be set (both can be set as
well). It is also fine if other KeyUsage bits are set.
```

The following flow describes the logic for the certificate validation when key usage is employed.

- If no KU extension is present, continue.
- If a KU is present but doesn't include digitalSignature or nonrepudiation, reject the certificate.
- If none of the above is true, continue.

> **Note**
>
> When IKEv2 is used with Cisco AnyConnect connections require that if a server certificate contains Key Usage, the attributes must contain digitalSignature AND (KeyAgreement OR KeyEncipherment).

Extended Key Usage (EKU) is another X509v3 extension defined in RFC3280, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile." This field indicates one or more purposes for which the certified public key may be used, in addition to or in place of the basic purposes indicated in the key usage extension field.

When Cisco AnyConnect is used, if the server certificate contains an EKU, the attributes must contain serverAuth (for SSL and IPsec) or ikeIntermediate (for IPsec only). Note that server certificates are not required to have a KU or an EKU to be accepted.

For certificate-based authentication, the FlexVPN server and Microsoft Windows 7 client certificates must have an Extended Key Usage (EKU) field as follows:

For the client certificate, EKU field = client authentication certificate.

For the server certificate, EKU field = server authentication certificate.

Debugging Authentication Using PKI

The following example illustrates the IKEv2 debug messages that are seen on the responder when the initiator attempts to validate itself by using a certificate that isn't configured on any trustpoints. We see that the presented certificate cannot be validated against any locally configured trustpoints.

Click here to view code image

```
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Retrieving trustpoint(s) from received
certificate hash(es)
IKEv2:(SA ID = 1):[PKI -> IKEv2] Retrieved trustpoint(s): NONE
IKEv2:% Received cert hash is invalid, using configured trustpoints from profile
for signing

IKEv2:(SA ID = 1):[IKEv2 -> PKI] Getting cert chain for the trustpoint 1
IKEv2:(SA ID = 1):[PKI -> IKEv2] Getting of cert chain for the trustpoint PASSED
IKEv2:(SESSION ID = 1,SA ID = 1):Get peer's authentication method
IKEv2:(SESSION ID = 1,SA ID = 1):Peer's authentication method is 'RSA'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Validating certificate chain
IKEv2:(SA ID = 1):[PKI -> IKEv2] Validation of certificate chain FAILED
IKEv2:(SESSION ID = 1,SA ID = 1):: Platform errors
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):Verify cert failed
IKEv2:(SESSION ID = 1,SA ID = 1):Verification of peer's authentication data FAILED
IKEv2:(SESSION ID = 1,SA ID = 1):Sending authentication failure notify
IKEv2:(SESSION ID = 1,SA ID = 1):Building packet for encryption.
Payload contents:
 NOTIFY(AUTHENTICATION_FAILED)
```

When an initiator attempts to authenticate itself, the trustpoint that is used must be configured within the IKEv2 profile using the **pki trustpoint** command. If no trustpoints are configured, then authentication will fail.

If the client has authenticated the trustpoint but not enrolled to it (and obtained a certificate), then authentication will fail. Even if the trustpoint is configured within the IKEv2 profile, it can only be used once enrolled.

Assuming that the initiator has a trustpoint configured, it will look to sign the AUTH payload with the private key related to the certificate contained in the trustpoint; this is then sent to the peer.

Certificate Expiry

Even if the local certificate has expired, it is still used for authentication. It is the responsibility of the receiver of the authentication payload to validate that the certificate is not expired.

If the responder should receive a certificate from the initiator that is not within the notBefore and notAfter lifetimes (expired), then IKE will fail. If IKEv2 debugging is enabled, the log messages show in the following example will be seen.

Click here to view code image

```
IKEv2:Current time is more than cert validity time
IKEv2:(SESSION ID = 32,SA ID = 1):Failed to set IKE policy: Failed to build
certificate payload
IKEv2:(SESSION ID = 32,SA ID = 1):Verification of peer's authentication data
FAILED
IKEv2:(SESSION ID = 32,SA ID = 1):Sending authentication failure notify
IKEv2:(SESSION ID = 32,SA ID = 1):Building packet for encryption.
Payload contents:
 NOTIFY(AUTHENTICATION_FAILED)
```

To ensure that the time is accurate, use Network Time Protocol (NTP). The command **show ntp associations** will list all configured NTP servers.

The PKI functions for rollover and re-enrollment can be affected if there is no authoritative time source. For this reason, you should always configure authenticated NTP on all of the routers that perform PKI functions.

The following example illustrates a client with an NTP server configured and synchronized. The asterisk (*) denotes that the peer is synced.

Click here to view code image

```
FlexClient#show ntp associations

  address         ref clock       st   when   poll reach  delay  offset   disp
*~10.10.10.1      127.127.1.1      8     17     64     1  0.000   0.000 189.44
 * sys.peer, # selected, + candidate, - outlyer, x falseticker, ~ configured
```

NTP has a number of debugging options, including clock adjustment behavior. The following example

illustrates NTP packet debugging, in which NTP packets can be seen being sent and received to/from the NTP server.

Click here to view code image

```
FlexClient#debug ntp packet
NTP packets debugging is on

NTP message sent to 10.10.10.1, from interface 'Ethernet0/0' (10.10.20.3).
NTP message received from 10.10.10.1 on interface 'Ethernet0/0' (10.10.20.3).
```

NTP uses the User Datagram Protocol (UDP) and uses port 123 for communication. In the following example, we see that the NTP server is not synchronized. After NTP packet debugging is enabled, we see that two-way communication fails between the client and the NTP server.

Click here to view code image

```
FlexClient#show ntp associations

  address         ref clock       st   when   poll reach  delay  offset   disp
 ~10.10.10.1      .INIT.          16      -     64     0  0.000   0.000 15937.
 * sys.peer, # selected, + candidate, - outlyer, x falseticker, ~ configured

FlexClient#debug ntp packet
NTP packets debugging is on

NTP message sent to 10.10.10.1, from interface 'NULL' (0.0.0.0).
```

Note

If a certificate expires and connectivity is still required, a trustpoint can be configured to ignore expired certificates by entering the **match certificate** command with the allow expired-certificates option within the trustpoint configuration. The following example illustrates the application of a certificate map to allow expired certificates. The configuration for the certificate map is not shown. Note that this is contrary to security best practice, but it may be required in certain situations.

Click here to view code image

```
crypto pki trustpoint pki
match certificate cert_map allow expired-certificate
```

Matching Peer Using Certificate Maps

If certificate maps are used to anchor a presented certificate to an IKEv2 profile, the attributes within the certificate must be matched within the certificate map.

The following example illustrates the output from IKEv2 debugging of a successful connection when a certificate map is used to anchor a client connection to an IKEv2 profile.

Click here to view code image

```
IKEv2:(SESSION ID = 39,SA ID = 1):Searching policy based on peer's identity
'hostname=flexclient.cisco.com' of type 'DER ASN1 DN'
IKEv2:Optional profile description not updated in PSH
IKEv2:Searching Policy with fvrf 0, local address 10.10.10.1
IKEv2:Using the Default Policy for Proposal
IKEv2:Found Policy 'default'
```

```
IKEv2:Found matching IKEv2 profile 'default'
```

The following example illustrates output from IKEv2 debugging when an unsuccessful connection occurs when a certificate map is used to anchor a client connection to an IKEv2 profile. Note the "failed to locate an item in the database" message.

Click here to view code image

```
IKEv2:(SESSION ID = 40,SA ID = 1):Searching policy based on peer's identity
'hostname=flexclient.cisco.com' of type 'DER ASN1 DN'
IKEv2:% IKEv2 profile not found
IKEv2:(SESSION ID = 40,SA ID = 1):: Failed to locate an item in the database
IKEv2:(SESSION ID = 40,SA ID = 1):Verification of peer's authentication data
FAILED
IKEv2:(SESSION ID = 40,SA ID = 1):Sending authentication failure notify
IKEv2:(SESSION ID = 40,SA ID = 1):Building packet for encryption.
```

The certificate map can be viewed within the running configuration by using the command shown in the following example.

Click here to view code image

```
Router#show running-config | section certificate map
crypto pki certificate map certmap 10
 subject-name eq hostname=flexclient.cisco.com
```

The IKEv2 profile must have a match statement within it; this is indicated by the message seen in the following example when there is an attempt to generate a new IKEv2 profile.

Click here to view code image

```
Router(config)#crypto ikev2 profile test
IKEv2 profile MUST have:
   1. A local and a remote authentication method.
   2. A match identity or a match certificate or match any statement.
```

Certificate Revocation

If revocation checking is enabled for the trustpoint, then the IKEv2 subsystem will make calls to the PKI subsystem. The PKI component will perform the revocation checking, which involves downloading a CRL or performing an OCSP request, depending on the configuration.

The following example illustrates a certificate being received when IKEv2 and PKI validation debugging has been enabled with the commands **debug crypto ikev2** and **debug crypto pki** validation respectively.

Click here to view code image

```
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Retrieving trustpoint(s) from received
certificate hash(es)
IKEv2:(SA ID = 1):[PKI -> IKEv2] Retrieved trustpoint(s): 'pki'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Getting cert chain for the trustpoint pki
IKEv2:(SA ID = 1):[PKI -> IKEv2] Getting of cert chain for the trustpoint PASSED
IKEv2:(SESSION ID = 5,SA ID = 1):Get peer's authentication method
IKEv2:(SESSION ID = 5,SA ID = 1):Peer's authentication method is 'RSA'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Validating certificate chain
IKEv2:(SA ID = 1):[PKI -> IKEv2] Validation of certificate chain FAILED
IKEv2:(SESSION ID = 5,SA ID = 1):
%PKI-6-PKI_CRL_DOWNLOADED: CRL download notification sent for Issuer = cn=sub.
IKEv2:(SESSION ID = 5,SA ID = 1):Save pubkey
```

```
IKEv2:(SESSION ID = 5,SA ID = 1):Verify peer's authentication data
IKEv2:[IKEv2 -> Crypto Engine] Generate IKEv2 authentication data
IKEv2:[Crypto Engine -> IKEv2] IKEv2 authentication data generation PASSED
IKEv2:(SA ID = 1):[IKEv2 -> Crypto Engine] Verify signed authentication data
IKEv2:(SA ID = 1):[Crypto Engine -> IKEv2] Verification of signed authentication
data PASSED
```

It can be seen that although the validation of the certificate chain fails because the CRL was not downloaded, the CRL is then downloaded and authentication is successful.

The following example shows the debug messages when there is an attempt to download a CRL; however the validation fails, and its failure generates an authentication failure notify payload.

Click here to view code image

```
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Getting cert chain for the trustpoint pki
IKEv2:(SA ID = 1):[PKI -> IKEv2] Getting of cert chain for the trustpoint PASSED
IKEv2:(SESSION ID = 8,SA ID = 1):Get peer's authentication method
IKEv2:(SESSION ID = 8,SA ID = 1):Peer's authentication method is 'RSA'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Validating certificate chain
IKEv2:(SA ID = 1):[PKI -> IKEv2] Validation of certificate chain FAILED
IKEv2:(SESSION ID = 8,SA ID = 1):
Router#
IKEv2:: Negotiation context locked currently in use
Router#
%PKI-3-SOCKETSELECT: Failed to select the socket.
IKEv2:(SESSION ID = 8,SA ID = 1):Verify cert failed
IKEv2:(SESSION ID = 8,SA ID = 1):Verification of peer's authentication data FAILED
IKEv2:(SESSION ID = 8,SA ID = 1):Sending authentication failure notify
IKEv2:(SESSION ID = 8,SA ID = 1):Building packet for encryption.
Payload contents:
 NOTIFY(AUTHENTICATION_FAILED)
```

Further investigation is required to determine why the verification failed. PKI debugging can be enabled to determine why the PKI component failed. The following example illustrates the messages generated after PKI debugging has been enabled.

Click here to view code image

```
CRYPTO_PKI: (A0011) Requesting CRL from SCEP
CRYPTO_PKI: locked trustpoint pki, refcount is 2
CRYPTO_PKI: http connection opened via VRF OUTSIDE
CRYPTO_PKI: Sending HTTP message

CRYPTO_PKI: Reply HTTP header:
HTTP/1.0
Host: 10.10.30.3

Router#
IKEv2:: Negotiation context locked currently in use
Router#
%PKI-3-SOCKETSELECT: Failed to select the socket.
```

It can be seen that an HTTP request is sent out, but the reply is not received. The message “failed to select the socket” indicates this.

To investigate further, TCP debug messages are enabled with the **debug ip tcp** command. The following example shows the debug messages generated. The messages indicate that the TCP socket is being opened, however the TCP SYN does not receive a reply.

Click here to view code image

```
tcp0: O CLOSED 10.10.30.3:80 10.10.10.1:48085 seq 3276603876
        OPTS 4 SYN  WIN 4128
tcp0: R SYNSENT 10.10.30.3:80 10.10.10.1:48085 seq 3276603876
        OPTS 4 SYN  WIN 4128
TCP0: timeout #1 - timeout is 4000 ms, seq 3276603876
TCP: (48085) -> 10.10.30.3(80)
```

In this case a firewall was blocking access to the CRL Distribution Point (CDP). Other explanations that would cause the same symptoms are; the incorrect CDP could be configured in a certificate, or the HTTP daemon on the server hosting the CDP could be malfunctioning. In these scenarios, debugging HTTP would be more prudent.

The **show crypto pki counters** command can be used to see counters relating to PKI behavior on the device. In the following example, it can be seen that a number of CRL fetches failed.

Click here to view code image

```
R1#show crypto pki counters
PKI Sessions Started: 50
PKI Sessions Ended: 50
PKI Sessions Active: 0
Successful Validations: 4
Failed Validations: 46
Bypassed Validations: 0
Pending Validations: 0
CRLs checked: 56
CRL - fetch attempts: 49
CRL - failed attempts: 45
CRL - rejected busy fetching: 1
AAA authorizations: 0
```

The output from debugging a successful authentication using PKI is shown in the following example. Note that the "PASSED" message indicates that the device successfully performed the cryptographic functions.

Click here to view code image

```
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Retrieving trustpoint(s) from received
certificate hash(es)
IKEv2:(SA ID = 1):[PKI -> IKEv2] Retrieved trustpoint(s): 'pki'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Getting cert chain for the trustpoint pki
IKEv2:(SA ID = 1):[PKI -> IKEv2] Getting of cert chain for the trustpoint PASSED
IKEv2:(SESSION ID = 200,SA ID = 1):Get peer's authentication method
IKEv2:(SESSION ID = 200,SA ID = 1):Peer's authentication method is 'RSA'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Validating certificate chain
IKEv2:(SA ID = 1):[PKI -> IKEv2] Validation of certificate chain PASSED
IKEv2:(SESSION ID = 200,SA ID = 1):Save pubkey
IKEv2:(SESSION ID = 200,SA ID = 1):Verify peer's authentication data
IKEv2:[IKEv2 -> Crypto Engine] Generate IKEv2 authentication data
IKEv2:[Crypto Engine -> IKEv2] IKEv2 authentication data generation PASSED
IKEv2:(SA ID = 1):[IKEv2 -> Crypto Engine] Verify signed authentication data
IKEv2:(SA ID = 1):[Crypto Engine -> IKEv2] Verification of signed authentication
data PASSED
```

Trustpoint Configuration

When digital signatures are used for authentication, one or more trustpoints can be defined within the IKEv2 profile. This is used to define which trustpoint should be used for authentication.

A trustpoint can be configured just for signing the local authentication payload by including the **sign** keyword, as illustrated in the following example.

Click here to view code image

```
Router(config-ikev2-profile)#pki trustpoint 1 sign
```

Or it can be configured just for verification of the remote peer by including the **verify** keyword, as illustrated here:

Click here to view code image

```
Router(config-ikev2-profile)#pki trustpoint 1 verify
```

If neither keyword is included, both signing and verification are performed with the same trustpoint.

Trustpoint Selection

When only a single trustpoint is configured on a device, there can be no discrepancy between peers as to which trustpoint is used. If more than one trustpoint is configured, then the CERTREQ payload will be used to give an indication of what PKI to use.

The following example illustrates the debug messages seen on an initiator when the received CERTREQ payload within the IKE_SA_INIT message did not match any locally configured trustpoints in the IKEv2 profile. In this instance, the first trustpoint from the IKEv2 profile will be used.

On the initiator (received IKE_SA_INIT)

Click here to view code image

```
IKEv2:(SESSION ID = 1,SA ID = 1):Processing IKE_SA_INIT message
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Retrieving trustpoint(s) from received
certificate hash(es)
IKEv2:(SA ID = 1):[PKI -> IKEv2] Retrieved trustpoint(s): NONE
IKEv2:% Received cert hash is invalid, using configured trustpoints from profile
for signing
```

The following example illustrates the debug messages seen on a responder when the received CERTREQ payload within the IKE_AUTH message did not match any locally configured CA certificates.

Click here to view code image

```
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Retrieving trustpoint(s) from received
certificate hash(es)
IKEv2:(SA ID = 1):[PKI -> IKEv2] Retrieved trustpoint(s): NONE
IKEv2:% Received cert hash is invalid, using configured trustpoints from profile
for signing

IKEv2:(SA ID = 1):[IKEv2 -> PKI] Getting cert chain for the trustpoint 1
IKEv2:(SA ID = 1):[PKI -> IKEv2] Getting of cert chain for the trustpoint PASSED
IKEv2:(SESSION ID = 1,SA ID = 1):Get peer's authentication method
IKEv2:(SESSION ID = 1,SA ID = 1):Peer's authentication method is 'RSA'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Validating certificate chain
```

```
IKEv2:(SA ID = 1):[PKI -> IKEv2] Validation of certificate chain FAILED
IKEv2:(SESSION ID = 1,SA ID = 1):: Platform errors
IKEv2:(SESSION ID = 1,SA ID = 1):Verify cert failed
IKEv2:(SESSION ID = 1,SA ID = 1):Verification of peer's authentication data FAILED
IKEv2:(SESSION ID = 1,SA ID = 1):Sending authentication failure notify
IKEv2:(SESSION ID = 1,SA ID = 1):Building packet for encryption.
Payload contents:
 NOTIFY(AUTHENTICATION_FAILED)
```

On receipt of the IKE_AUTH packet, the responder will match the exchange to an IKEv2 profile and then perform cryptographic checks on the presented certificate.

If there are multiple IKEv2 profiles, ensure that there are no overlapping match statements within profiles. An IKEv2 profile that is associated with a tunnel interface should be used only by that tunnel interface; however, when a device is acting as a responder, the selection of the IKEv2 profile is based on the match condition within the IKEv2 profile. The profile that is configured on the tunnel interface must match the profile that is selected using the peer's IKE identity. If this does not match, the negotiation will be aborted.

Should both IKEv2 profiles have overlapping match statements and the incorrect profile is selected, the debug messages within the following example will be seen. In this case, two tunnel interfaces with different tunnel destination addresses and IPsec and IKEv2 profiles are configured. Although IKE traffic is received from the correct tunnel destination IP address, the incorrect IKEv2 profile is matched (which is not applied to the IPsec profile protecting the tunnel interface), as the IKEv2 profile is used to match the peer and determine the protection the other IKEv2 profile will be used.

Click here to view code image

```
map_db_check_isakmp_profile profile did not match
map_db_find_best did not find matching map
IPSEC(ipsec_process_proposal): proxy identities not supported
IKEv2:(SA ID = 1):There was no IPSEC policy found for received TS
IKEv2:(SA ID = 1):Sending TS unacceptable notify
```

The IKE system will make calls to the PKI system; from a simplistic perspective the PKI system will validate that the certificate used has not expired and check that the AUTH payload was cryptographically signed with the private key that relates to the public key contained within the certificate.

Pre-Shared Key

When pre-shared keys are used for authentication, each peer can use a shared secret to authenticate itself to the peer, or the peer to itself. The keys are held within the keyring, which is a database of identities with associated keys. The identity can be populated with the same value as an IKE identity, but as the following section describes, this is not mandatory. The keys must match between peers. For the initiator, the identity within the keyring must be configured with either an IP address or hostname. The responder must have the identity within the keyring set to IP address or IKE identity.

The keyring can be configured locally or can be obtained from RADIUS using AAA.

The local or remote keywords can be included when the key is defined. This allows asymmetrical keys to be defined with different keys on each peer. If the configuration uses asymmetrical keys, ensure that the local key on one peer matches the remote key on the other peer.

The only method to verify that the local keyring has been configured correctly is the **show running-**

configuration command. In the following example we see a single IKEv2 keyring configured.

Click here to view code image

```
Routert#show running-config | section ikev2 keyring
crypto ikev2 keyring key1
 peer 10.10.10.1
  address 10.10.10.1
  pre-shared-key cisco
```

If RADIUS is used, a symmetrical key can be defined on the RADIUS server, or asymmetrical keys can be configured. Table 14-6 illustrates the different methods of configuration for pre-shared keys when RADIUS is used.

Attribute	Type	Format	Configuration
`Tunnel-Password`	IETF	String (attribute ID 69)	`Tunnel-Password=string`
`ipsec:ikev2-password-local`	Cisco AV Pair	String	`cisco-avpair="ipsec:ikev2-password-local=string"`
`ipsec:ikev2-password-remote`	Cisco AV Pair	String	`cisco-avpair="ipsec:ikev2-password-remote=string"`

Table 14-6 *RADIUS PSK Configuration*

If a keyring that is configured within an IKEv2 profile is used and does not have the identity of the peer defined, the debug message shown in the following example will be seen.

Click here to view code image

```
IKEv2:% Getting preshared key from profile keyring key1
IKEv2:% key not found.
IKEv2:Failed to initiate sa
```

> **Note**
>
> If an IP address is used as the IKEv2 identity, as the identity within the keyring can be configured as a subnet, ensure that the presented identities will always reside within one of the configured subnets.

When AAA is used with RADIUS to obtain the pre-shared key, the RADIUS server must be reachable, and the username used as the identity for the lookup must be configured on the RADIUS server. The following section describes troubleshooting AAA with RADIUS.

The name mangler function can be used with AAA to obtain the identity. If the name mangler is used, be sure to verify that it is correctly configured to extract the relevant part of the identity that is used as the lookup. The following example illustrates the IKEv2 debug messages where a presented identity type of flexclient.cisco.com of type FQDN is seen. The hostname (flexclient) is extracted and then used as the lookup. In this scenario, ensure that there is an account with the username flexclient on the RADIUS server.

Click here to view code image

```
IKEv2:(SESSION ID = 12,SA ID = 1):Peer's authentication method is 'PSK'
```

```
IKEv2:(SESSION ID = 12,SA ID = 1):Get peer's preshared key for flexclient.cisco.
com
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Password request sent
IKEv2:IKEv2 local AAA - user author request for 'flexclient'
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received password response
```

Cisco IOS provides the ability to configure the pre-shared key as a hexadecimal value. If this method is used, ensure that the corresponding peer has the same shared secret configured. This can be verified by using an ASCII-to-HEX conversion tool.

The following example illustrates a configuration in which one peer has a pre-shared key configured using the ASCII value of "cisco" and its peer has the equivalent hexadecimal value.

Click here to view code image

```
Router1#show run | section ikev2 keyring
crypto ikev2 keyring key1
 peer 10.10.10.1
  address 10.10.10.1
  pre-shared-key cisco

Router2#show run | section ikev2 keyring
crypto ikev2 keyring key1
 peer 10.10.10.100
  address 10.10.10.100
  pre-shared-key hex 636973636f
```

Extensible Authentication Protocol (EAP)

It is very common for architectures of large-scale deployments of VPN clients to use a backend database to manage the user accounts. In this case, EAP is a common authentication method that allows a RADIUS server to store the client credentials and perform the authentication. When EAP is used, the FlexVPN server will pass through the client credentials and allow the EAP server on the RADIUS device to make the accept or reject decision based on the attributes supplied by the client.

When EAP is used, the client must authenticate the VPN headend (and responder) using digital signatures, so a certificate must be configured on the FlexVPN server. The FlexVPN server certificate must have an Extended Key Usage (EKU) of server-auth.

The EKU can be verified by examining the certificate. The following example illustrates how to verify the EKU with the **show crypto pki certificate verbose** command.

Click here to view code image

```
Router#show crypto pki certificates verbose
...
X509v3 extensions:
    X509v3 Key Usage: 86000000
      Digital Signature
      Key Cert Sign
      CRL Signature
   X509v3 extensions:
  X509v3 Extended Key Usage:
    TLS Web Server Authentication
    TLS Web Client Authentication
```

When IKEv2 is used with EAP, the components are the client, the FlexVPN server, and the RADIUS server. The client and FlexVPN server will communicate over IKEv2; however, EAP messages are

exchanged within the IKE_AUTH exchange. The FlexVPN server and RADIUS server communicate using RADIUS, where the EAP messages are transported within the EAP Message attribute.

Figure 14-2 illustrates the components of the EAP exchange, with the protocol flows.

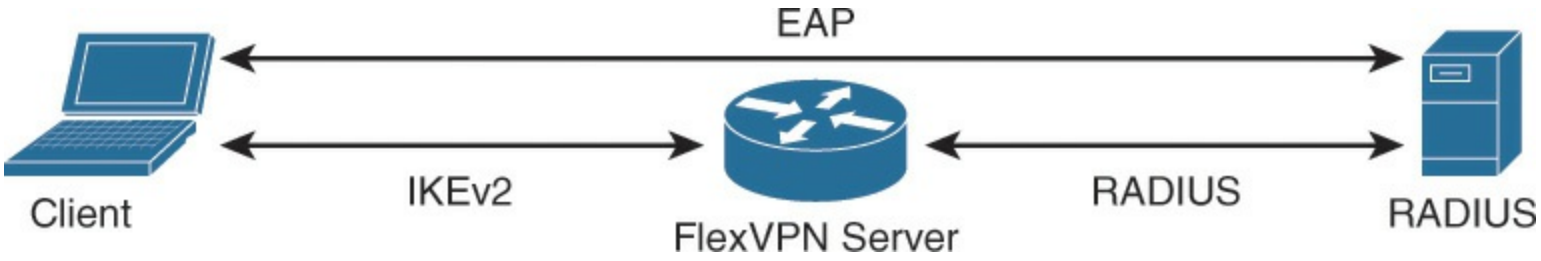


Figure 14-2 *IKEv2 with EAP*

The RADIUS server configuration can be verified by examining the running configuration, as shown in the following example.

Click here to view code image

```
Router#show run | section radius
aaa group server radius radius_group
 server-private 10.10.20.1 auth-port 1812 acct-port 1813 key cisco123
 aaa authentication login rad1 group radius_group
```

RADIUS connectivity can be verified with the **test aaa** command, as illustrated in the following example. RADIUS debugging was enabled to highlight the connectivity to the RADIUS server and verify that the username (test) and password (lab) presented to the RADIUS server was successfully authenticated.

Click here to view code image

```
router#test aaa group radius_group test lab new-code

RADIUS/ENCODE(00000000):Orig. component type = INVALID RADIUS/ENCODE(00000000):
dropping service type, "radius-server attribute 6 on-for-login-auth" is off
RADIUS(00000000): Config NAS IP: 10.10.10.1
RADIUS(00000000): sending
RADIUS/ENCODE: Best Local IP-Address 10.10.10.1 for Radius-Server 10.10.20.1
RADIUS(00000000): Send Access-Request to 10.10.20.1:1812 id 1812/1, len 50
RADIUS:  authenticator CA DB F4 9B 7B 66 C8 A9 - D1 99 4E 8E A4 46 99 B4
RADIUS:  User-Password       [2]   18  *
RADIUS:  User-Name           [1]   6   "test"
RADIUS:  NAS-IP-Address      [4]   6   10.10.10.1
RADIUS: Received from id 1812/1 10.10.20.1:1812, Access-Accept, len 44
```

Assuming that the client initiates the IKEv2 session and the IKE_SA_INIT exchange was successful, the EAP messages will be exchanged in the IKE_AUTH exchange. To denote that EAP is being used, the client excludes the AUTH payload in the first IKE_AUTH request, at this point the FlexVPN server will prove its own identity by sending its own AUTH payload.

The following example illustrates the output that was generated from debugging IKEv2 on a FlexVPN client when EAP is used; it shows the IKE_AUTH request without the AUTH payload.

Click here to view code image

```
IKEv2:(SESSION ID = 1,SA ID = 1):Building packet for encryption.
Payload contents:
 VID IDi CERTREQ NOTIFY(HTTP_CERT_LOOKUP_SUPPORTED) CFG SA TSi TSr
NOTIFY(INITIAL_CONTACT) NOTIFY(USE_TRANSPORT_MODE) NOTIFY(SET_WINDOW_SIZE)
NOTIFY(ESP_TFC_NO_SUPPORT) NOTIFY(NON_FIRST_FRAGS)
```

If the client is Cisco IOS, then the IKE identity must *not* be configured as an IP address or Distinguished name (DN); in these cases the connection will fail, and the following debug message will be seen on the FlexVPN server.

Click here to view code image

```
IKEv2:Cannot use IP address as EAP identity
IKEv2:Received response from authenticator
IKEv2:(1): Authenticator sent NULL EAP message
IKEv2:(1): Extensible Authentication Protocol failed
IKEv2:(1): Auth exchange failed
IKEv2:(1): Abort exchange
```

Some third-party peers, such as Windows 7 and above, will send the identity as an IP address that will be rejected by the FlexVPN server. For peers that send the identity as an IP address, it is necessary to configure **query-identity** so that the headend sends an EAP Identity Request to the client during IKE_AUTH. The EAP identity is used later for authorization requests; note that even if we receive an EAP identity from the client, it can still be overridden by the EAP server once the EAP authentication has completed.

Figure 14-3 illustrates the behavior of IKEv2 on Cisco IOS without **query-identity** enabled. The IKE identity is used in the EAP-Response.

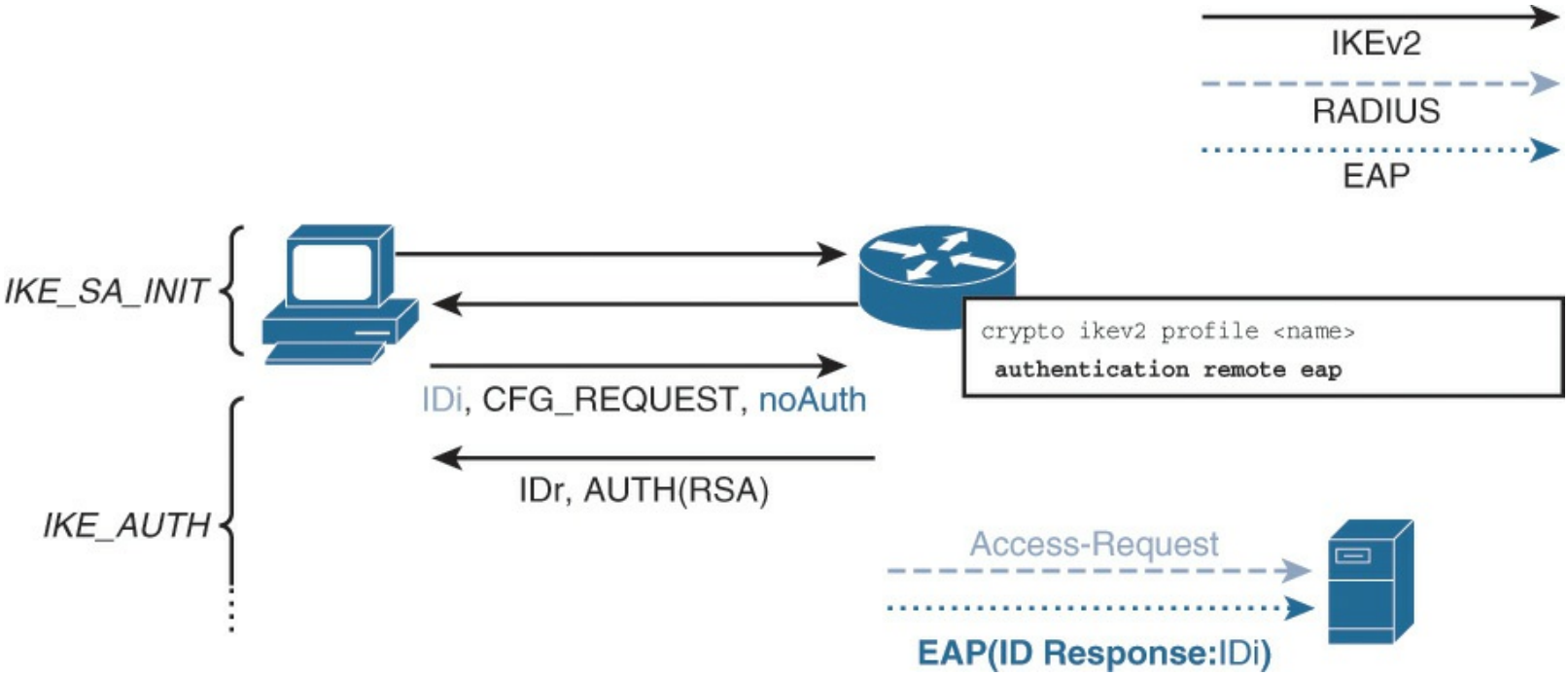


Figure 14-3 *EAP without the **query-identity** Command*

Figure 14-4 illustrates the behavior of IKEv2 on Cisco IOS with **query-identity** enabled. The IKE identity is not used in the EAP-Response. The FlexVPN server will query the client for the identity within an EAP-Request which is then used.

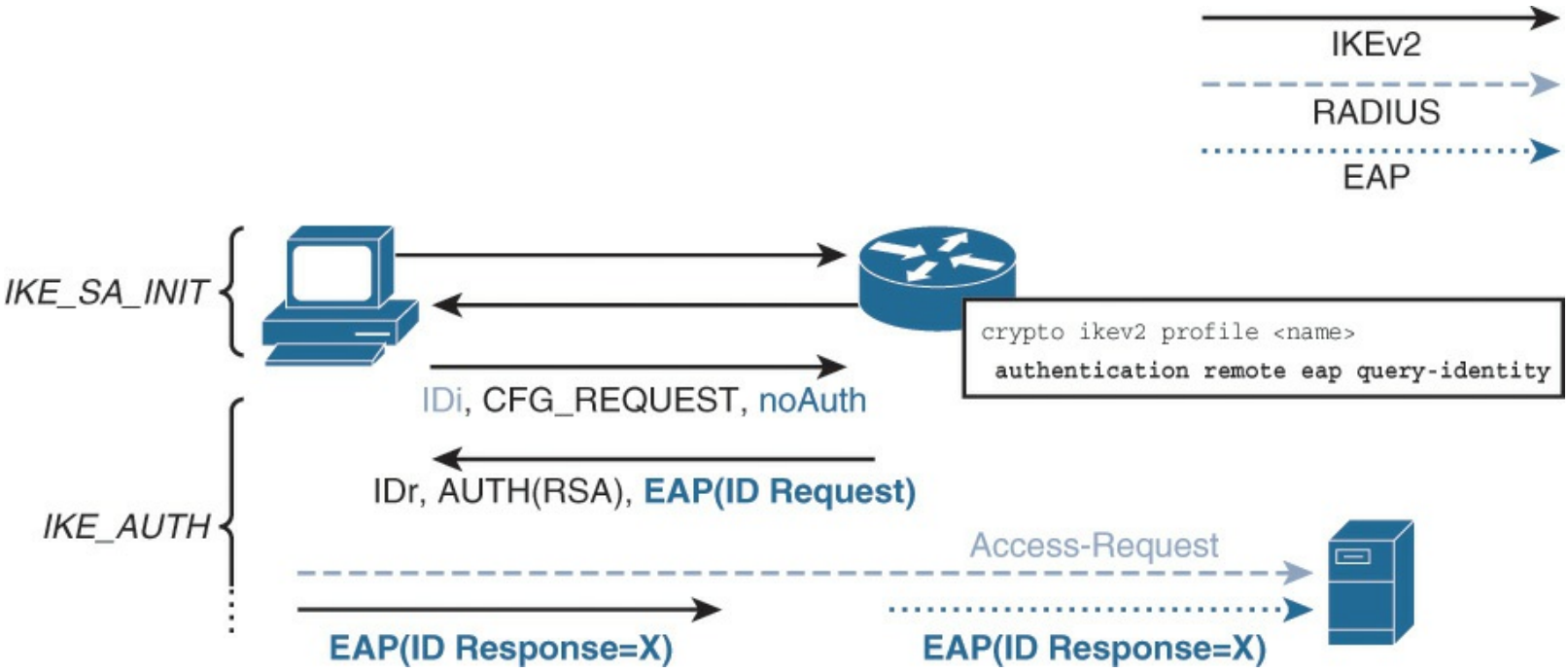


Figure 14-4 *EAP with the **query-identity** Command*

On receipt of the IKE_AUTH, the FlexVPN server will process the identity. The following example illustrates a debug message taken from a client using MS-CHAPv2 that shows the identity, which in hexadecimal is 636973636F, but in ASCII is "cisco." The same identity is then used as the EAP User-Name.

Click here to view code image

```
IKEv2:Exchange type: IKE_AUTH, flags: INITIATOR
IKEv2:Message id: 0x2, length: 68
REAL Decrypted packet:
    00 00 00 0E 02 3B 00 0A 01 63 69 73 63 6F
EAP  Next payload: NONE, reserved: 0x0, length: 14
Code: response: id: 59, length: 10
Type: identity
```

The FlexVPN server will relay the identity to the RADIUS server within the Access-Request message. The following IKEv2 and RADIUS debug messages show a RADIUS request with the username of "cisco," which was used as the IKEv2 identity.

Click here to view code image

```
IKEv2:(1): Processing EAP response
RADIUS/ENCODE(00000013):Orig. component type = VPN IPSEC
RADIUS:  AAA Unsupported Attr: service-type      [345] 4   1 RADIUS/
ENCODE(00000013): dropping service type, "radius-server attribute 6
on-for-login-auth" is off
RADIUS(00000013): Config NAS IP: 0.0.0.0
RADIUS(00000013): Config NAS IPv6: ::
RADIUS/ENCODE(00000013): acct_session_id: 9
RADIUS(00000013): sending
RADIUS/ENCODE: Best Local IP-Address 10.10.10.10.1 for Radius-Server 10.10.10.100
RADIUS(00000013): Send Access-Request to 10.10.10.10.100:1812 id 1645/10, len 82
RADIUS:  authenticator 10 16 B6 71 69 BE 3C 8F - D4 9B 77 8B AF 5A 9E ED
RADIUS:  Service-Type        [6]   6   Login                    [1] RADIUS:
Calling-Station-Id  [31]  13  "10.10.20.100"
RADIUS:  User-Name           [1]   7   "cisco"
RADIUS:  EAP-Message         [79]  12
RADIUS:   02 3B 00 0A 01 63 69 73 63 6F            [ ;cisco]
```

```
RADIUS:  Message-Authenticato[80]  18   RADIUS:   4F DE 44 97 7B 04 0E A2 C9 FC 55
FA 72 2F 53 34          [ OD{Ur/S4]
RADIUS:  NAS-IP-Address      [4]   6   10.10.10.1
RADIUS(00000013): Sending a IPv4 Radius Packet
```

Once the Access-Challenge is received on the FlexVPN server from the RADIUS server, it will be relayed via IKEv2 back to the client. This is illustrated in the following example.

Click here to view code image

```
RADIUS: Received from id 1812/30 10.10.10.100:1812, Access-Challenge, len 98
RADIUS:  authenticator 02 E3 BC 5C 7B EF DC 38 - 0C 14 12 4A 4F A0 6C 08
RADIUS:  Framed-IP-Address   [8]   6   172.16.1.1
RADIUS:  Service-Type        [6]   6   Framed                    [2]
RADIUS:  Framed-Protocol     [7]   6   PPP                       [1]
RADIUS:  EAP-Message         [79]  24
RADIUS:   01 3C 00 16 04 10 67 30 CF 35 9B 2E A0 56 85 F4 51 7B 0F 01 7F 6B
[ <g05.VQ{k]
RADIUS:  Message-Authenticato[80]  18
RADIUS:   E1 11 71 E1 DC 3A DB 30 89 3D 62 A8 F0 FC 63 FB            [ q:0=bc]
RADIUS:  State               [24]  18
RADIUS:   16 47 B3 FF 16 7B B7 6A 0B 77 42 E6 F7 FC 0A 7B            [ G{jwB{]
RADIUS(00000013): Received from id 1811/30
RADIUS/DECODE: EAP-Message fragments, 22, total 22 bytes
RADIUS: Constructed " ppp negotiate"
IKEv2:(SA ID = 1):[AAA -> IKEv2] Successful response received
IKEv2:Received response from authenticator
IKEv2:(SESSION ID = 5,SA ID = 1):Generating EAP request
IKEv2:(SESSION ID = 5,SA ID = 1):Building packet for encryption.
Payload contents:
 EAP

IKEv2:(SESSION ID = 5,SA ID = 1):Sending Packet [To 10.10.10.10:500/From
10.10.10.1:500/VRF i0:f0]
Initiator SPI : CEC6599A1DDEDAD4 - Responder SPI : 7044B33369C2FC4A Message id: 2
IKEv2 IKE_AUTH Exchange RESPONSE
Payload contents:
 ENCR

...
IKEv2:(SA ID = 1):[AAA -> IKEv2] Successful response received
IKEv2:Received response from authenticator
IKEv2:(SESSION ID = 5,SA ID = 1):Generating EAP request
IKEv2:(SESSION ID = 5,SA ID = 1):Building packet for encryption.
Payload contents:
 EAP
```

If the **timeout** keyword is used on the FlexVPN server when EAP is configured as the remote authentication method, as the following example illustrates, then the FlexVPN server must receive a reply to the EAP messages sent in the IKE_AUTH exchange within the timeout period.

Click here to view code image

```
authentication remote eap timeout <seconds>
```

When a remote access client is used with EAP, user authorization is enabled by default if AAA user authentication is configured. When the command **aaa authorization user** is enabled with the **cached** attribute, the RADIUS attributes that are received for the user account during the authentication phase, such as the assigned IP address for the client, are automatically applied to the client's connection.

> **Note**
>
> With EAP, user authorization is only possible with an external RADIUS server and not a locally configured user.

Debugging should not be focused only on the FlexVPN server. If the client has the incorrect EAP type configured, then this can be viewed by the logs on the RADIUS server.

The connectivity between the FlexVPN server and the RADIUS server must be stable, and the shared secret that is used for RADIUS authentication must be configured on both the FlexVPN server and the RADIUS server.

Authorization

When a client requests attributes via a configuration payload, authorization must be enabled on the FlexVPN server. The attributes that are sent to the peer by a configuration payload can be obtained either locally via the IKEv2 authorization policy or via RADIUS.

The username used to make the lookup as to which policy should be applied can be obtained by using the peer's IKE identity, or by using the name mangler function to extract a part of the presented identity.

When local authorization is used, all attributes are stored locally, and no external connectivity is required. The IKEv2 system will make calls to the AAA system, which will then return the attributes contained within the IKEv2 authorization policy.

When attributes that are used for authorization are obtained via RADIUS, the IKE system will query the AAA system, which in turn will make the RADIUS requests.

When authorization is enabled, the identity lookup is performed after the authentication phase has completed.

The following example shows IKEv2 and AAA authorization debug messages from a FlexVPN server using local authorization. This illustrates the authentication completing and the authorization occurring. The IKEv2 system makes a call to the AAA system, which returns the authorization response containing the attributes.

Click here to view code image

```
IKEv2:(SA ID = 1):[IKEv2 -> Crypto Engine] Verify signed authentication data
IKEv2:(SA ID = 1):[Crypto Engine -> IKEv2] Verification of signed authentication
data PASSED
IKEv2:(SESSION ID = 17,SA ID = 1):Processing INITIAL_CONTACT
IKEv2:Using mlist default and username service1 for group author request
AAA/BIND(0000002B): Bind i/f
AAA/AUTHOR (0x2B): Pick method list 'default'
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorisation request sent
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response
IKEv2:(SESSION ID = 17,SA ID = 1):Received valid config mode data
```

If local authorization is used and RADIUS services are not required to be enabled (**`no aaa new-model`** and local aaa authorization list), the local keyword can be used within the IKEv2 profile configuration when configuring local authorization, as illustrated in the following example.

Click here to view code image

```
aaa authorization group cert list default default local
```

When authorization is used, if the client requests an IP address via configuration payload and the FlexVPN server cannot allocate this, then authentication will fail. The request for an IP address must be fulfilled, and so this request will cause the authentication to fail if an address cannot be allocated. The following example shows the debug message that will be seen if an IP address cannot be allocated.

Click here to view code image

```
IKEv2:Error constructing config reply
IKEv2:(SESSION ID = 31,SA ID = 1):Error in retrieving config mode data to send
IKEv2:(SESSION ID = 31,SA ID = 1):Fail to alloc IP addr
```

If authorization is used to apply raw configuration commands to a virtual-access interface, the syntax of the command is not checked before it is applied. If the syntax of a command is incorrect and would not be accepted by the CLI, the IKE session will fail. The following example illustrates this behavior. The configuration that is applied to a virtual-access interface via authorization is incorrect. The letter "a" is missing from the word "access-group," which will result in the IKE session failing.

Click here to view code image

```
aaa attribute list service1
 attribute type interface-config "ip ccess-group service1 in" protocol ip
```

This incorrect configuration being applied generates the following IKEv2 debug messages.

Click here to view code image

```
IKEv2:AAA clone request failed, error 7
IKEv2:(SESSION ID = 0,SA ID = 1):Error while applying per user attributes
```

Be mindful when applying configuration via AAA authorization to the virtual-access interface. The order of the commands must be the same as when applying manually. For example if a virtual-access interface is assigned to a VRF with the command **vrf forwarding**, this command must be applied before an IP address is applied to the interface, because the application of the **vrf forwarding** command will remove any configured IP addresses.

The following example shows which debug commands can be used to give a verbose output when AAA authorization is used.

```
debug aaa authorization
debug aaa attr
```

To determine the username that is used for the authorization lookup, AAA protocol debugging can be enabled with the **debug aaa proto {local | radius}** command.

The following example illustrates how this debugging is enabled along with IKEv2 debugging; the authorization policy named "service1" is used.

Click here to view code image

```
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorisation request sent
AAA/LOCAL/AUTHEN: starting
AAA/LOCAL/AUTHEN(17): authorizing crypto service1
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response
IKEv2:(SESSION ID = 7,SA ID = 1):Received valid config mode data
IKEv2:Config data received:
```

RADIUS combines both authentication and authorization within the Access-Accept message. When using authentication and authorization, separate message exchanges will occur between the FlexVPN server and the RADIUS server. If user authorization is enabled for PSK or EAP, the cached keyword can be included within the aaa configuration to cache the RADIUS attributes received in the PSK retrieval or EAP success respectively. If it is required to apply attributes received from RADIUS to user authorization for PSK or EAP, then ensure that the cached keyword is included within the AAA authorization configuration, as illustrated in the following example.

Click here to view code image

```
crypto ikev2 profile default
 aaa authorization user {psk|eap} cached
```

The creation of the virtual-access interface can be debugged with the command **debug vtemplate**. This command has a number of options, with **cloning**, **event**, and **error** being more relevant when it is necessary to debug issues when authorization is used to apply configuration.

The following example shows the messages that were generated when **debug vtemplate cloning** was enabled and a dynamically generated virtual-access interface was created; aaa authorization was enabled where the commands applied to the virtual-access interface can be seen.

Click here to view code image

```
VT[Vi1]:Applying config commands on process "VTEMPLATE Background Mgr" (373)
 VT[Vi1]:default ip mtu 1420
 VT[Vi1]:default ip tcp adjust-mss 1400
 VT[Vi1]:default ip access-group subscriber1 in
 VT[Vi1]:default ip verify unicast source reachable-via rx
 VT[Vi1]:default ip unreachables
 VT[Vi1]:default ip unnumbered loopback1
 VT[Vi1]:default ip access-group service1 in
 VT[Vi1]:end
 VT[Vi1]:Remove cloneblk AAA from vaccess with vtemplate/AAA
 VT[Vi1]:Unclone vaccess, 3 command(s) to be removed
 VT[Vi1]:default tunnel protection ipsec profile default
 VT[Vi1]:default tunnel vrf OUTSIDE
 VT[Vi1]:default tunnel source Ethernet0/0
 VT[Vi1]:end
```

Data Encryption

Once the IPsec Security Association is created and data is transmitted and protected by IPsec, any issues with data encryption and integrity protection will obviously involve IPsec.

Debugging IPsec

When an IPsec Security Association is created, peers will negotiate what to protect by using traffic selectors and how this will be protected by using transforms that are contained within a proposal.

Although IKEv2 can perform a negotiation in which the most specific selectors are used between peers, this is relevant only when using crypto maps and dVTI (tunnel mode IPsec) and not for GRE or VTI interfaces.

To verify the IPsec transform set that is used, use the **show crypto IPsec transform-set** command. The following example illustrates this command being executed on a device with two transform sets configured.

Click here to view code image

```
Router#show crypto ipsec transform-set
Transform set default: { esp-aes esp-sha-hmac  }
   will negotiate = { Transport,  },

Transform set aes-sha256: { esp-aes esp-sha256-hmac  }
   will negotiate = { Transport,  },
```

The transform set must be referenced within the IPsec profile that is used to protect the intended interface. The following example illustrates viewing a default profile with the relevant transform set applied.

Click here to view code image

```
Router#show crypto ipsec profile
IPSEC profile default
     IKEv2 Profile: default
     Security association lifetime: 4608000 kilobytes/3600 seconds
     Responder-Only (Y/N): N
     PFS (Y/N): Y
     DH group:  group14
     Mixed-mode : Disabled
     Transform sets={
          aes-sha256:  { esp-aes esp-sha256-hmac  } ,
```

If the initiator sends an IPsec transform that does not match on the responder, the responder will fail to find a matching policy. The following example shows IKEv2 debugging output from a responder that received a transform that it did not support. It can be seen that in addition to the authentication data, a NO_PROPOSAL_CHOSEN notify message is sent in return.

Click here to view code image

```
IKEv2:(SESSION ID = 473,SA ID = 1):Set received config mode data
IKEv2:(SESSION ID = 473,SA ID = 1):Processing IKE_AUTH message
IKEv2:% DVTI create request sent for profile default with PSH index 1.
IKEv2:(SESSION ID = 473,SA ID = 1):
%LINEPROTO-5-UPDOWN: Line protocol on Interface Virtual-Access1, changed state
to down
IKEv2:% DVTI Vi1 created for profile default with PSH index 1.
IKEv2:IPSec policy validate request sent for profile default with psh index 1.
IKEv2:(SA ID = 1):[IPsec -> IKEv2] Callback received for the validate proposal -
FAILED.
IKEv2:(SESSION ID = 473,SA ID = 1):Received Policies: : Failed to find a matching
policyESP: Proposal 1:  3DES SHA256 Don't use ESN
IKEv2:(SESSION ID = 473,SA ID = 1):Expected Policies: : Failed to find a matching
policy
IKEv2:(SESSION ID = 473,SA ID = 1):: Failed to find a matching policy
IKEv2:(SESSION ID = 473,SA ID = 1):Sending no proposal chosen notify
***
IKEv2:(SESSION ID = 473,SA ID = 1):Building packet for encryption.
Payload contents:
 VID IDr CERT AUTH NOTIFY(NO_PROPOSAL_CHOSEN)
```

On the initiator, the IKEv2 debugs can be slightly confusing. The AUTH payload is received from the responder, with the notify payload indicating that no proposal was chosen. Because this is an IKE_AUTH payload, the IKEv2 profile is firstly matched, this can be seen where the authentication messages indicate successful authentication. After the successful authentication the IKE session is created; however, because no proposal was chosen, the IKE SA is deleted and a DELETE message is

sent to the peer to indicate that the IKE session has been torn down.

Click here to view code image

```
IKEv2:(SESSION ID = 1,SA ID = 1):Received Packet [From 10.10.10.1:500/To
10.10.50.5:500/VRF i0:f0]
Initiator SPI : 58C298669EA6A95F - Responder SPI : 38F270737BD3EE39 Message id: 1
IKEv2 IKE_AUTH Exchange RESPONSE
Payload contents:
 VID IDr CERT AUTH NOTIFY(NO_PROPOSAL_CHOSEN)

IKEv2:(SESSION ID = 1,SA ID = 1):Process auth response notify
...
IKEv2:(SESSION ID = 1,SA ID = 1):Verify peer's policy
IKEv2:(SESSION ID = 1,SA ID = 1):Peer's policy verified
IKEv2:(SESSION ID = 1,SA ID = 1):Get peer's authentication method
IKEv2:(SESSION ID = 1,SA ID = 1):Peer's authentication method is 'RSA'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Validating certificate chain
IKEv2:(SA ID = 1):[PKI -> IKEv2] Validation of certificate chain PASSED
IKEv2:(SESSION ID = 1,SA ID = 1):Save pubkey
IKEv2:(SESSION ID = 1,SA ID = 1):Verify peer's authentication data
IKEv2:[IKEv2 -> Crypto Engine] Generate IKEv2 authentication data
IKEv2:[Crypto Engine -> IKEv2] IKEv2 authentication data generation PASSED
IKEv2:(SA ID = 1):[IKEv2 -> Crypto Engine] Verify signed authentication data
IKEv2:(SA ID = 1):[Crypto Engine -> IKEv2] Verification of signed authentication
data PASSED
IKEv2:(SESSION ID = 1,SA ID = 1):Check for EAP exchange
IKEv2:Using mlist default and username default for group author request
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorisation request sent
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Close PKI Session
IKEv2:(SA ID = 1):[PKI -> IKEv2] Closing of PKI Session PASSED
IKEv2:(SESSION ID = 1,SA ID = 1):IKEV2 SA created; inserting SA into database.
SA lifetime timer (86400 sec) started
IKEv2:(SESSION ID = 1,SA ID = 1):Session with IKE ID PAIR (hostname=R1.cisco.com,
hostname=R5.cisco.com,cn=r5,o=cisco,ou=service1) is UP
IKEv2:IKEv2 MIB tunnel started, tunnel index 1
IKEv2:(SESSION ID = 1,SA ID = 1):Checking for duplicate IKEv2 SA
IKEv2:(SESSION ID = 1,SA ID = 1):No duplicate IKEv2 SA found
IKEv2:(SESSION ID = 1,SA ID = 1):Queuing IKE SA delete request reason: unknown
IKEv2:(SESSION ID = 1,SA ID = 1):Sending DELETE INFO message for IPsec SA [SPI:
0xAEF5E4BE]
IKEv2:(SESSION ID = 1,SA ID = 1):Building packet for encryption.
Payload contents:
 DELETE
```

An IPsec Security Association will fail to be established if the tunnel interface types do not match. VTI does not use any form of encapsulation, and GRE interfaces use GRE; these are incompatible. The following example illustrates the IKEv2 debug output that was generated on a responder that was configured with a GRE interface when the initiator was configured with a VTI interface. The TS unacceptable message indicates that the interface types are incompatible. The authentication and authorization (if configured) will be successful; however, because the IPsec policy does not match the locally configured policy, the IKE session will fail.

Click here to view code image

```
IKEv2:(SESSION ID = 1067,SA ID = 1):Processing IKE_AUTH message
IKEv2:% DVTI create request sent for profile default with PSH index 1.
IKEv2:(SESSION ID = 1067,SA ID = 1):
%LINEPROTO-5-UPDOWN: Line protocol on Interface Virtual-Access1, changed state
```

```
to down
IKEv2:% DVTI Vi1 created for profile default with PSH index 1.
IKEv2:IPSec policy validate request sent for profile default with psh index 1.
IKEv2:(SA ID = 1):[IPsec -> IKEv2] Callback received for the validate proposal -
FAILED.
IKEv2:(SESSION ID = 1067,SA ID = 1):: There was no IPSEC policy found for
received TS
IKEv2:(SESSION ID = 1067,SA ID = 1):Sending TS unacceptable notify
```

The following example shows the output that was generated by using the **debug crypto ikev2 packet debugging** command. This output illustrates the IPsec proposal; the traffic selectors can be seen, which indicate all traffic being protected, and GRE is not used, as indicated by the **proto id** being 0.

Click here to view code image

```
SA  Next payload: TSi, reserved: 0x0, length: 44
  last proposal: 0x0, reserved: 0x0, length: 40
  Proposal: 1, Protocol id: ESP, SPI size: 4, #trans: 3    last transform: 0x3,
reserved: 0x0: length: 12
    type: 1, reserved: 0x0, id: AES-CBC
    last transform: 0x3, reserved: 0x0: length: 8
    type: 3, reserved: 0x0, id: SHA256
    last transform: 0x0, reserved: 0x0: length: 8
    type: 5, reserved: 0x0, id: Don't use ESN
 TSi  Next payload: TSr, reserved: 0x0, length: 24
    Num of TSs: 1, reserved 0x0, reserved 0x0
    TS type: TS_IPV4_ADDR_RANGE, proto id: 0, length: 16
    start port: 0, end port: 65535
    start addr: 0.0.0.0, end addr: 255.255.255.255
 TSr  Next payload: NOTIFY, reserved: 0x0, length: 24
    Num of TSs: 1, reserved 0x0, reserved 0x0
    TS type: TS_IPV4_ADDR_RANGE, proto id: 0, length: 16
    start port: 0, end port: 65535
    start addr: 0.0.0.0, end addr: 255.255.255.255
```

IPsec Anti-Replay

IPsec provides anti-replay protection against the duplication of encrypted packets or generation of traffic outside of the sequence number window by assigning a monotonically increasing sequence number to each encrypted packet. The receiving IPsec endpoint maintains a list of which packets it has already processed on the basis of these numbers with the use of a sliding window of all acceptable sequence numbers.

Figure 14-5 illustrates the behavior of the anti-replay window. Encapsulation Security Payload packets with a sequence number outside of the window will be discarded.

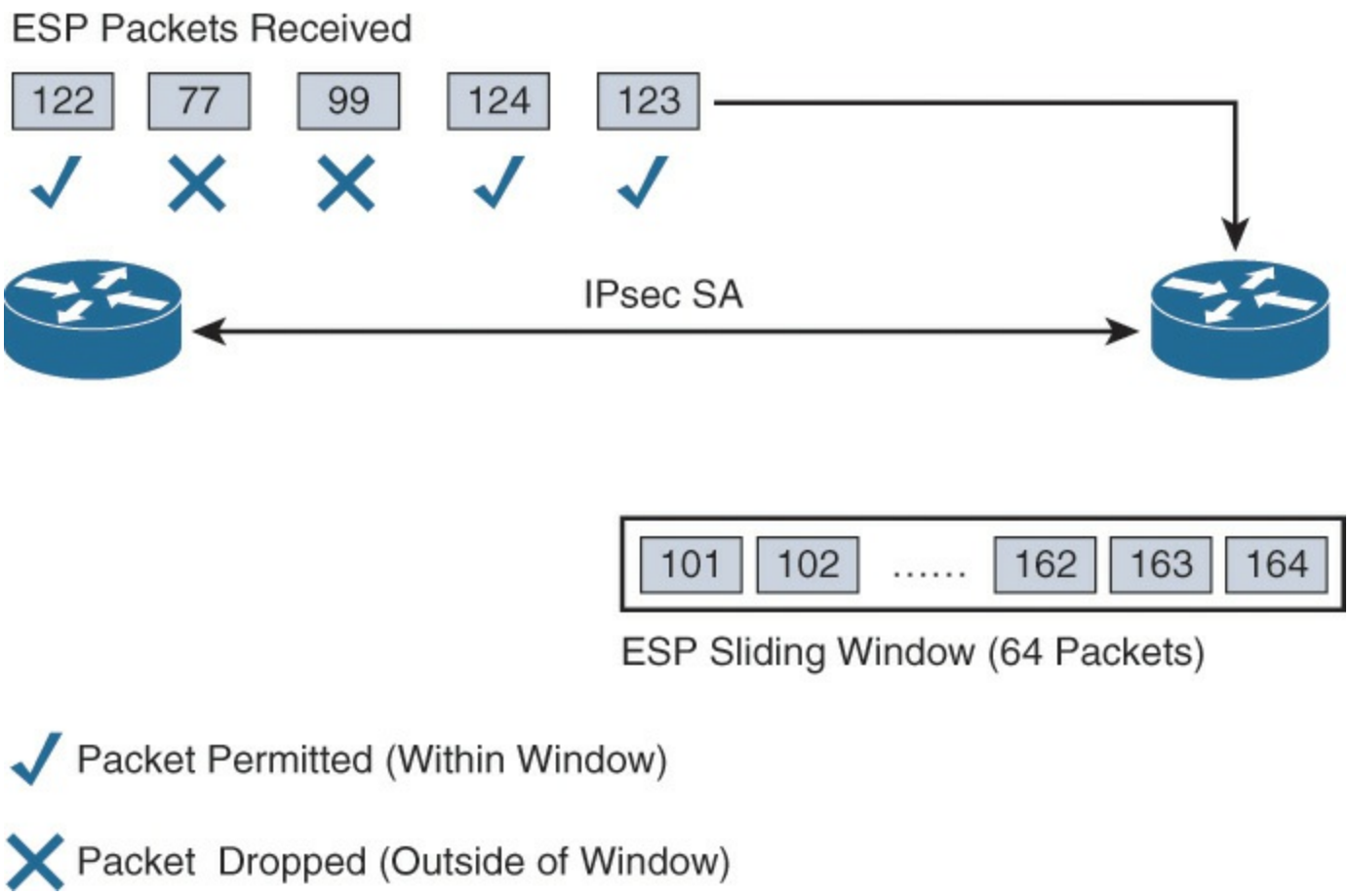


Figure 14-5 *Anti-Replay Window*

If a packet is received outside of the sliding window, the message illustrated in the following example is generated.

Click here to view code image

```
%CRYPTO-4-PKT_REPLAY_ERR: decrypt: replay check failed connection id=#, sequence
number=#
```

This message can be generated because of malicious or nonmalicious behavior. If an attacker captured encrypted Encapsulation Security Payload packets and replayed these, the message would be generated. Likewise, a message would be generated if an attacker sent random Encapsulation Security Payload packets that could not be verified. The message can also occur if traffic comes out of order, if parallel paths exist on the transport network, or if QoS is applied and results in some packets being queued. Unequal packet processing paths can occur if large IPsec packets that require IP reassembly before decryption are delayed while smaller packets are passed in a system under load.

The following example illustrates a method for determining the IPsec Security Association that was the owner of a packet failing the anti-replay check. The connection ID is 771, with the packet having a sequence number of 1193.

Click here to view code image

```
%CRYPTO-4-PKT_REPLAY_ERR: decrypt: replay check failed
        connection id=771, sequence number=1193
```

The IPsec Security Association Database can be viewed to determine which IPsec Security Association relates to connection ID 771.

Click here to view code image

```
Router#show crypto ipsec sa | in peer|conn id
   current_peer 10.1.1.1 port 500
        conn id: 771, flow_id: SW:771, sibling_flags 80000046, crypto-map:
Tunnel0-head-0
        conn id: 772, flow_id: SW:772, sibling_flags 80000046, crypto-map:
Tunnel0-head-0
```

On devices running IOS-XE since 3.7 / 15.2(4)S, details of the IPsec Security Association are included in the syslog message, as shown below.

Click here to view code image

```
%IPSEC-3-REPLAY_ERROR: IPSec SA receives anti-replay error, DP Handle 6,
          src_addr 10.10,10.1, dest_addr 10.10.10.2, SPI 0x1a2b3c4d
```

This message does not show the Encapsulation Security Payload sequence number, unlike IOS, which will show the sequence number. On the ASR if the sequence number is required, the ASR datapath packet tracing feature can be used to capture traffic that is transmitted over the dataplane.

The following illustrates the method to enable the ASR datapath packet tracing feature to determine the Encapsulation Security Payload sequence number of dropped traffic due to the anti-replay check.

Step 1. Set up the platform conditional debugging filter in order to match traffic from the peer device.

Click here to view code image

```
Router#debug platform condition ipv4 10.2.0.200/32 ingress
Router#debug platform condition start
```

Step 2. Enable packet tracing with the copy option in order to copy the packet header information. The size is set to 100 because only the Encapsulation Security Payload header is required to determine the sequence number.

Click here to view code image

```
Router#debug platform packet enable
Router#debug platform packet-trace packet 64
Router#debug platform packet-trace copy packet input l3 size 100
```

Step 3. When anti-replay errors are detected, use the packet trace buffer to identify the packet dropped due to replay, and the Encapsulation Security Payload sequence number can be found in the packet copied.

Click here to view code image

```
Router#show platform packet-trace summary
Pkt    InputOutput   State  Reason
0 Gi1/0/0  Tu1  CONS   Packet Consumed
1 Gi1/0/0  Tu1  CONS   Packet Consumed
2 Gi1/0/0  Tu1  DROP   053 (IpsecInput)
3 Gi1/0/0  Tu1  DROP   053 (IpsecInput)
4 Gi1/0/0  Tu1  CONS   Packet Consumed
```

Step 4. The previous output shows that packet numbers 2 and 3 were dropped, so they should be examined further.

Click here to view code image

```
Router#show platform packet-trace packet 2
```

```
Packet: 2   CBUG ID: 2
Summary
  Input : GigabitEthernet1/0/0
  Output: Tunnel1
  State : DROP 053 (IpsecInput)
  Timestamp : 3233497953773
Path Trace
  Feature: IPV4
Source  : 10.1.1.1
Destination : 10.1.1.2
Protocol: 50 (ESP)
  Feature: IPSec
Action: DECRYPT
SA Handle : 3
SPI   : 0x4c1d1e90
Peer Addr : 10.1.1.1
Local Addr: 10.1.1.2
  Feature: IPSec
Action: DROP
Sub-code  : 019 - CD_IN_ANTI_REPLAY_FAIL
Packet Copy In
  45000428 00110000 fc329575 0a0200c8 0a010064 4c1d1e90 000000df 790aa252
  e9951cd9 57024433 d97c7cb8 58e0c869 2101f1ef 148c2a12 f309171d 1b7a4771
  d8868af7 7bae9967 7d880197 46c6a079 d0143e43 c9024c61 0045280a d57b2f5e
  23f06bc3 ab6b6b81 c1b17936 98939509 7aec966e 4dd848d2 60517162 9308ba5d
```

The Encapsulation Security Payload sequence number has an offset of 24 that starts from the IP header, as emphasized in bold and italics in the previous output. In this particular example, the Encapsulation Security Payload sequence number for the dropped packet is 0xdf, which is 223 in decimal.

The default anti-replay window size in Cisco IOS is 64 packets; however this is configurable to a maximum of 1024 packets by using the command **set security-association replay window-size {64 | 128 | 256 | 512 | 1024 }** within the IPsec profile or by using the global command **crypto IPsec security-association replay window-size {64 | 128 | 256 | 512 | 1024 }.**

To disable the anti-replay check, use the command **set security-association replay window-size disable** within the IPsec profile or use the global command **crypto IPsec security-association replay window-size disable**.

If IPsec traffic is received that is dropped by the anti-replay function and the anti-replay window is set as the default 64 packets, it is worthwhile to set it to the maximum of 1024 packets. If dropped traffic continues, then take packet captures of all messages at both hosts to fully diagnose the cause. By viewing the packet captures, the packet flow can be examined to see what traffic is being sent out of order or injected into the flow.

Data Encapsulation

The GRE protocol is stateless and only encapsulates traffic within the GRE header. This results in very little room for error or the chance of untoward behavior occurring.

Mismatching GRE Tunnel Keys

If GRE tunnel keys are used and these are not aligned between peers, then the IPsec Security Association will be created. The GRE tunnel key is not negotiated, but it is a locally configured value that is added to the GRE header of the outgoing packets.

Once the IPsec Security Association forms, traffic will be sent into the tunnel interface, which is protected by IPsec. At the peer device when traffic is decrypted and de-encapsulated, if the tunnel keys do not match the packet will be discarded.

This can be a tricky issue to diagnose, for the IPsec Security Association will indicate that traffic has been encapsulated by the sender and de-encapsulated by the receiver; however, IP connectivity will fail over the tunnel interface.

A simple method to troubleshoot this issue is to view the configuration on the tunnel interface to ensure that the key is the same on both peers.

The output in the following example illustrates how to confirm the configured GRE tunnel key.

Click here to view code image

```
Router#show interface tunnel 1
Tunnel1 is up, line protocol is down
  Hardware is Tunnel
  ...
 interface <OK>
  Tunnel protocol/transport GRE/IP
    Key 0x2, sequencing disabled
    Checksumming of packets disabled
```

Overlay Routing

Once an IPsec Security Association is successfully created, IP packets can flow over it. The Routing Information Base (RIB) must have entries to the destination prefixes for all traffic that is to be sent into the tunnel. The RIB can be populated manually with static routes, IKEv2 routing, or a dynamic routing protocol. Manually configuring routing is not dynamic unless it is integrated with enhanced object tracking; however it offers minimal overhead. IKEv2 routing allows for prefixes to be distributed between peers without the overhead that is required in a routing protocol. Routing protocols allow dynamic routing decisions to occur over the tunnel, but it come with the cost of running the routing protocol and keepalives.

Static Routing

When static routing is used with the next hop as the tunnel interface, the route will be installed in the RIB only when the tunnel is in the up/up state. The following example illustrates a configuration that will only install the 192.1680.0/16 prefix into the RIB only if the tunnel interface is in the up state.

Click here to view code image

```
ip route 192.168.0.0 255.255.0.0 Tunnel1
```

With this configuration, you are at the mercy of the tunnel; if IKEv2 DPD is not enabled, then if the tunnel peers lose connectivity the tunnel can be in the up state, even though there is no IP connectivity between peers.

IKEv2 Routing

If IKEv2 routing is used, then prefixes are exchanged in the initial IKE authentication exchange via the configuration payload. These prefixes stay statically installed in the RIB for the duration of the IKEv2 SA and can be changed only when a new IKEv2 session is created.

If a FlexVPN client has concurrent IKEv2 SAs to multiple FlexVPN servers and IKEv2 routing is used, if prefixes are added or removed on the FlexVPN servers a new IKEv2 SA is required to make the client aware of these changes.

When IKEv2 routing is used, the prefixes are obtained locally from the IKEv2 authorization policy or via AAA and RADIUS. Ensure that the correct prefixes are used and that these are obtainable when AAA occurs.

Third-party remote access clients, which do not use split tunneling commonly, do not require a default route sent via the configuration payload, although this is implementation specific.

If FlexVPN clients are used with IKEv2 routing, ensure that both the client and server advertise the tunnel interface IP address via configuration payload. This is achieved by using aaa authorization and the **route set interface** command being enabled within the IKEv2 authorization policy.

The following example illustrates a FlexVPN server that has a number of connected clients. The clients are assigned IP addresses via a configuration payload. One of the clients is unreachable; although it was allocated an IP address, this was not installed into the RIB, which results in loss of connectivity.

Click here to view code image

```
Router#show crypto session brief
Status: A- Active, U - Up, D - Down, I - Idle, S - Standby, N - Negotiating
        K - No IKE
ivrf = (none)
           Peer      I/F        Username          Group/Phase1_id   Uptime Status
     10.10.60.6      Vi2                 hostname=Router2.cisco.com,cn 00:01:01
UA
     10.10.50.5      Vi1                 hostname=Router1.cisco.com,cn 00:31:38
UA
```

For the peer router1.cisco.com, we see the IP address assigned via the configuration payload is 192.168.1.14.

Click here to view code image

```
Router#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
2         10.10.10.1/500        10.10.50.5/500        OUTSIDE/none         READY
      Encr: AES-CBC, keysize: 128, PRF: SHA256, Hash: SHA256, DH Grp:14, Auth
sign: RSA, Auth verify: RSA
      Life/Active Time: 86400/220 sec
      CE id: 1026, Session-id: 19
      Status Description: Negotiation done
      Local spi: B6D1DDD763BE26D7       Remote spi: 8639ADD29DBB80B2
      ...
      Assigned host addr: 192.168.1.14
      Initiator of SA : No
```

> **Note**
>
> We would expect to see the assigned IP address within the Remote subnets: field; however this field is not shown as the remote prefix was not installed.

Within the RIB there is no route for the 192.168.1.14 prefix.

Click here to view code image

```
Router#show ip route 192.168.1.14
% Subnet not in table
```

By debugging IKEv2, we see that the FlexVPN client did not update the IP address assigned to it via the configuration payload with the **config-set** command.

Click here to view code image

```
IKEv2:(SESSION ID = 1,SA ID = 1):Config-type: Config-set
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: app-version, length: 256, data:
Cisco IOS Software,
..
```

On the remote client the ability to update the peer of the assigned IP address had been removed, which can be seen from the following output, which is missing the **route set interface** statement.

Click here to view code image

```
Router1#show crypto ikev2 authorization policy
 IKEv2 Authorization Policy : default
  route accept any tag : 1 distance : 1
```

By default the prefixes installed when IKEv2 routing is used will use an administrative distance of 1. If the administrative distance has been changed with the **route accept** command within the IKEv2 authorization policy, ensure that no other prefixes exist with a lower administrative distance unless required.

If using the **route set local** command within the IKEv2 authorization policy to apply certain prefixes to the RIB for certain peers, ensure that the correct IP address is applied and reachable via the peer.

Dynamic Routing Protocols

When running routing protocols over an IPsec Security Association, the IP address of the tunnel interface will be used to source and receive traffic pertaining to establishing and updating the routing protocol. The IP address of the peer must be reachable via the tunnel interface. The routing adjacency will form once the tunnel has been established and prefixes distributed between routing neighbors.

The following example illustrates a simple connectivity test between tunnel interfaces; the ping command is used to validate that the peer's IP address is reachable from the source of the local tunnel interface.

Click here to view code image

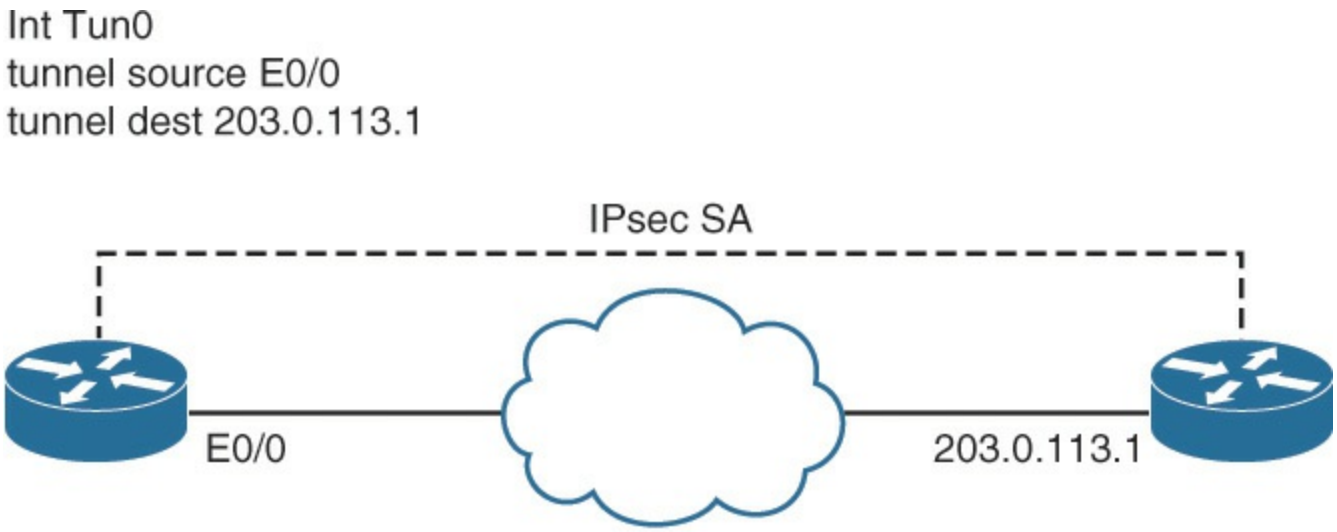
```
Router#ping 192.168.1.1 source tunnel 1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.1.19
```

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/5 ms
```

When routing protocols are used over a tunnel interface, ensure that the tunnel destination address is not advertised within the tunnel. Figure 14-6 illustrates that the tunnel destination is advertised from within the tunnel interface, which will result in traffic entering the tunnel and being encapsulated and protected by IPsec. The lookup for the tunnel destination address will be within the tunnel, which would result in traffic being encapsulated once more; however, at this point the packet will be dropped and the syslog message seen in the following example will be generated.

Click here to view code image

```
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up
%TUN-5-RECURDOWN: Tunnel0 temporarily disabled due to recursive routing
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to down
%LINEPROTO-5-UPDOWN:
```



Routing Table (RIB)

Destination	Next Hop and Interface
0.0.0.0/0	209.165.201.2 (Eth0/0)
10.0.0.0/30	Connected (Tun0)
10.10.210.0/24	10.0.0.2 (Tun0)
203.0.0.0/16	10.0.0.2 (Tun0)

Figure 14-6 *Recursive Routing*

To rectify the flapping tunnel interface, remove the offending prefix (203.0.0.0/16) that is advertised within the tunnel so that the tunnel destination is reachable only via the tunnel source interface and not within the tunnel itself.

Many routing protocols will have an initial handshake, during which both peers establish a neighborship and then exchange prefixes. The initial handshake usually requires packets of a relatively small size. When large amounts of prefixes are exchanged, the packet size will grow, and the packet size may then reach the maximum transmission unit size resulting in traffic being fragmented. If IPsec traffic between hosts is dropped because of fragmentation issues, it is common for a routing protocol to establish a neighborship; then if the routing updates fail to be passed between

peers, the neighborship will drop, resulting in flapping neighbors.

In this case, the IP MTU of the tunnel can be manually lowered with the **ip mtu** command, or the MTU of the tunnel can be automatically adjusted with PMTUD by using the **tunnel path-mtu-discovery** command, which is applied to the tunnel interface.

Summary

This chapter gives an understanding of how to undertake troubleshooting of IKE and IPsec. Knowing the operation of protocols will assist an engineer when troubleshooting issues with IPsec VPNs.

A key take away from this chapter is to understand the protocol flows of working architectures, so that when untoward behavior occurs, it can easily be spotted.

Following a defined methodology will allow the cause of the issue to be located. The steps listed in Table 14-1 allow a layered approach to troubleshooting to be conducted.

The following tables can be used as a reference for commands that can be used when diagnosing behavior and enabling debugging.

Table 14-7 lists useful **show** commands that can be used to verify IKEv2 and IPsec behavior.

Command	Purpose
show crypto ikev2 sa	Shows all active IKEv2 SAs
show crypto ikev2 sa detailed	Shows all active IKEv2 SAs with detailed information about the IKE SA
show crypto ikev2 session detail	Shows IKEv2 SA and corresponding child SA parameters
show crypto ikev2 stats	Shows IKEv2 negotiation stats, along with system resource limits
show crypto ikev2 stats exchange	Shows IKEv2 negotiation statistics for IKEv2 message types
show crypto ikev2 stats ext-service	Shows IKEv2 statistics for external service interaction
show crypto ikev2 diagnose error	Shows a history of IKEv2 error events with traceback recorded
show crypto ikev2 diagnose error <count>	Shows list of IKEv2 error counters
show crypto session detail	Shows same details as it would for IKEv1, shows uptime as well
show crypto IPsec sa detailed	Shows the IPsec Security Association database
show crypto sockets	Shows crypto sockets corresponding to virtual interfaces
show crypto ikev2 client flexvpn <profile-name> detail	Shows client side parameters and downloaded attributes from the flexvpn server

Table 14-7 *IKEv2 and IPsec* ***Show*** *Commands*

Table 14-8 lists useful **debug** commands that can be used to verify IKEv2 and IPsec behavior.

Command	Purpose
debug crypto condition <>	Conditional debugging so as to obtain debugs corresponding to a specific session.
debug crypto ikev2	Debugging IKEv2 exchanges
debug crypto ikev2 internal	IKEv2 internal debugging with events
debug crypto ikev2 error	Debug IKEv2 error conditions
debug crypto ikev2 packet	Unencrypted details of contents of ikev2 exchanges
debug crypto ikev2 client flexvpn	For flexvpn client troubleshooting
debug crypto kmi	Key message exchanges between IKEv2 and IPsec key engine
debug crypto IPsec	IPsec debugs

Table 14-8 *IKEv2 and IPsec* ***Debug*** *Commands*

Table 14-9 lists useful commands that can be used to verify interactions when AAA, RADIUS, and EAP are used.

Command	Purpose
debug aaa authentication	AAA authentication debugging
debug aaa authorization	AAA authorization debugging
debug aaa accounting	AAA accounting debugging
debug radius all	RADIUS debugging
debug eap all	EAP debugging

Table 14-9 *AAA, RADIUS and EAP* ***Debug*** *Commands*

Table 14-10 lists useful **show** commands that can be used to verify PKI operations.

Command	Purpose
show crypto pki certificates verbose	Shows verbose information contained within a certificate
show crypto pki timers	Shows timers relating to CA and CRLs
show crypto pki trustpoints status	Shows the status of all trustpoints
show crypto pki counters	Shows counters related to PKI functions
show crypto key mypubkey all/ec/rsa	Shows cryptographic keys
show crypto pki session	Shows active PKI sessions
show crypto pki crl	Shows downloaded CRL

Table 14-10 *PKI* ***Show*** *Commands*

Table 14-11 lists useful **debug** commands that can be used to verify PKI operations.

Command	Purpose
debug crypto pki certificate	Certificates used between client and server
debug crypto pki message	PKI message debugging (verbose) between client and server
debug crypto pki transactions	PKI transactions between client and server
debug crypto pki validation	PKI validation

Table 14-11 *PKI **Debug** Commands*

References

http://www.cisco.com/c/en/us/td/docs/security/vpn_client/anyconnect/anyconnect31/release/notes/any Release Notes for Cisco AnyConnect Secure Mobility Client, Release 3.1

http://www.cisco.com/c/en/us/support/docs/content-networking/adaptive-session-redundancy-asr/117858-technote-asr-00.html ASR Datapath Packet Tracing Feature

Part VII: IPsec Overhead

Chapter 15. IPsec Overhead and Fragmentation

Introduction

Finding out how much overhead IPsec will add to a given packet is not a simple task—there are many different reasons why overhead may be involved in producing an IPsec packet. In the first half of this chapter, we will first look at the structure of GRE, Encapsulation Security Payload, and Authentication Header protocols and the overhead involved and the characteristics of the algorithms that may be used to protect the traffic. We will then look at several ways to derive the plaintext maximum transmission unit (MTU) from the transport MTU and present a table and a set of formulas that make it easy to compute the maximum IPsec overhead for any given situation.

In the second half of this chapter, we will first review fragmentation in IPv4 and IPv6 as well as the concepts of MTU, Path MTU Discovery (PMTUD), and Maximum Segment Size (MSS) tuning. Then, we will look at fragmentation in IPsec and explain how the previous concepts translate to virtual interfaces. We will finish with recommendations on how to avoid the undesirable effects of fragmentation.

This chapter contains a lot of theory. Even though it would have been easier to produce a short chapter with a simple table of algorithm combinations and associated overhead, we strongly feel that it is crucial to build a deep understanding of these topics in order to avoid the many traps of IPsec fragmentation. Performance and packet drop problems due to incorrect MTU/MSS or broken PMTUD are far more common than they should be, and too many of those problems are due to lack of knowledge about how IPsec handles fragmentation.

Considerations regarding legacy crypto map configurations have been intentionally left out. Crypto maps introduce additional complexity with regard to fragmentation that we feel would only add confusion in an already dense chapter. Furthermore, as mentioned in Chapter 4 "IOS IPsec Implementation", the way forward in IOS VPN is clearly to move away from legacy crypto maps in favor of tunnel interfaces due to their numerous advantages.

Computing the IPsec Overhead

This section describes the various components involved in computing the IPsec overhead.

General Considerations

The following factors may contribute to the IPsec encapsulation overhead:

- the addition of a new IP header (tunnel mode only)
- the GRE encapsulation overhead (GRE over IPsec only)
- the addition of an Authentication Header or Encapsulation Security Payload header and trailer
- the payload structure of the encryption algorithm (Encapsulation Security Payload only)
- the length of the plaintext and the block size of the encryption algorithm (Encapsulation Security Payload only)
- the output size of the integrity algorithm (Encapsulation Security Payload with integrity or Authentication Header only)
- padding required for payload or integrity check value (ICV) alignment

- additional TFC padding

The IPsec-v3 standards (RFCs4301 to 4303) introduced new concepts such as combined-mode algorithms (which provide both confidentiality and integrity) and Traffic Flow Confidentiality (TFC). Encapsulation Security Payload allows for TFC to enable extra padding to hide the actual length of the plaintext in an Encapsulation Security Payload packet, which is not supported by Cisco IOS. TFC also allows for the generation of dummy packets to be sent that are used to shape encrypted traffic flows to hide the appearance of encrypted traffic. The following example illustrates enabling dummy traffic on Cisco IOS.

Click here to view code image

```
Router(config)#crypto ipsec security-association dummy pps ?
  <0-25>  Rate of simulated traffic (in PPS)

Router(config)#crypto ipsec security-association dummy seconds ?
  <1-3600>  Rate of simulated traffic (seconds between packets)
```

For the sake of simplicity and because IOS does not allow it, we will not cover TFC padding in this chapter; refer to RFC4303 for details.

Combined-mode algorithms (AES-GCM for example) provide confidentiality and integrity services. A combined-mode algorithm returns both a ciphertext and an integrity check value (ICV) as output, thus a separate integrity algorithm is not needed. Particular combined-mode algorithms (AES-GMAC for example) may provide no encryption and only integrity; those can be used with Encapsulation Security Payload to carry unencrypted but authenticated traffic.

A valid but rare scenario is the combination of Encapsulation Security Payload and Authentication Header in tunnel or transport mode (double encapsulation). Since that scenario can be reduced to having nested Encapsulation Security Payload and Authentication Header Security Associations, the overhead can be calculated in two steps provided that the order of encapsulation is known (that is, which transform is applied first). Refer to RFC2401 for detailed considerations regarding nested transforms.

IPsec Mode Overhead (without GRE)

Encapsulation Security Payload and Authentication Header in tunnel mode add an IP header in front of the original packet. The base length of an IP header is 20 bytes for IPv4 and 40 bytes for IPv6. The actual length can be greater if IP options (for IPv4) or extension headers (for IPv6) are added.

With IPv4, the protocol field in the new IP header is set to 50 for Encapsulation Security Payload or 51 for Authentication Header and the Encapsulation Security Payload or Authentication Header is next in the encapsulation chain. With IPv6, the Encapsulation Security Payload or Authentication Header is added as an extension header to the new IP packet.

Table 15-1 shows the IP header overhead of tunnel mode and transport mode (without GRE) for IPv4 and IPv6.

IPsec Mode (without GRE)		IP Header Overhead
Transport	over IPv4	None
	over IPv6	None
Tunnel	over IPv4	20 bytes + IP options (if any)
	over IPv6	40 bytes + non-IPsec extensions (if any)

Table 15-1 *IPsec Overhead*

GRE Overhead

GRE/IPsec (see Chapter 4, "IOS IPsec implementation") involves two successive encapsulations: first the original packet within GRE, then the resulting GRE packet within IPsec. During the first step, an IP header (IPv4 or IPv6 depending on the GRE endpoints) and a GRE header are added at the beginning of the plaintext packet. The length of the GRE header depends on the tunnel configuration (for example, presence of the Key field).

The influence of these two additional headers on the total IPsec overhead depends on the IPsec mode used in the second step:

- with transport mode, the IP header from GRE encapsulation is reused by IPsec, and only the GRE header becomes part of the plaintext data;
- with tunnel mode, the IP and GRE headers both become part of the plaintext data, and a new IP header is added by IPsec.

It is important to differentiate the number of bytes added to the IP header and to the plaintext, as the total length of the plaintext can have an influence on the amount of padding required to reach the appropriate block size for encryption (Encapsulation Security Payload with certain algorithms only; see "Encryption Overhead" later in this section). It is also important to remember that using GRE with tunnel mode brings a potentially unnecessary IP header that serves no purpose when using IPsec on Cisco IOS except for NAT traversal when there are multiple nodes behind the same PAT device with the same public address but unique private addresses.

Figure 15-1 shows the order of headers in the output packet for each of the two modes.

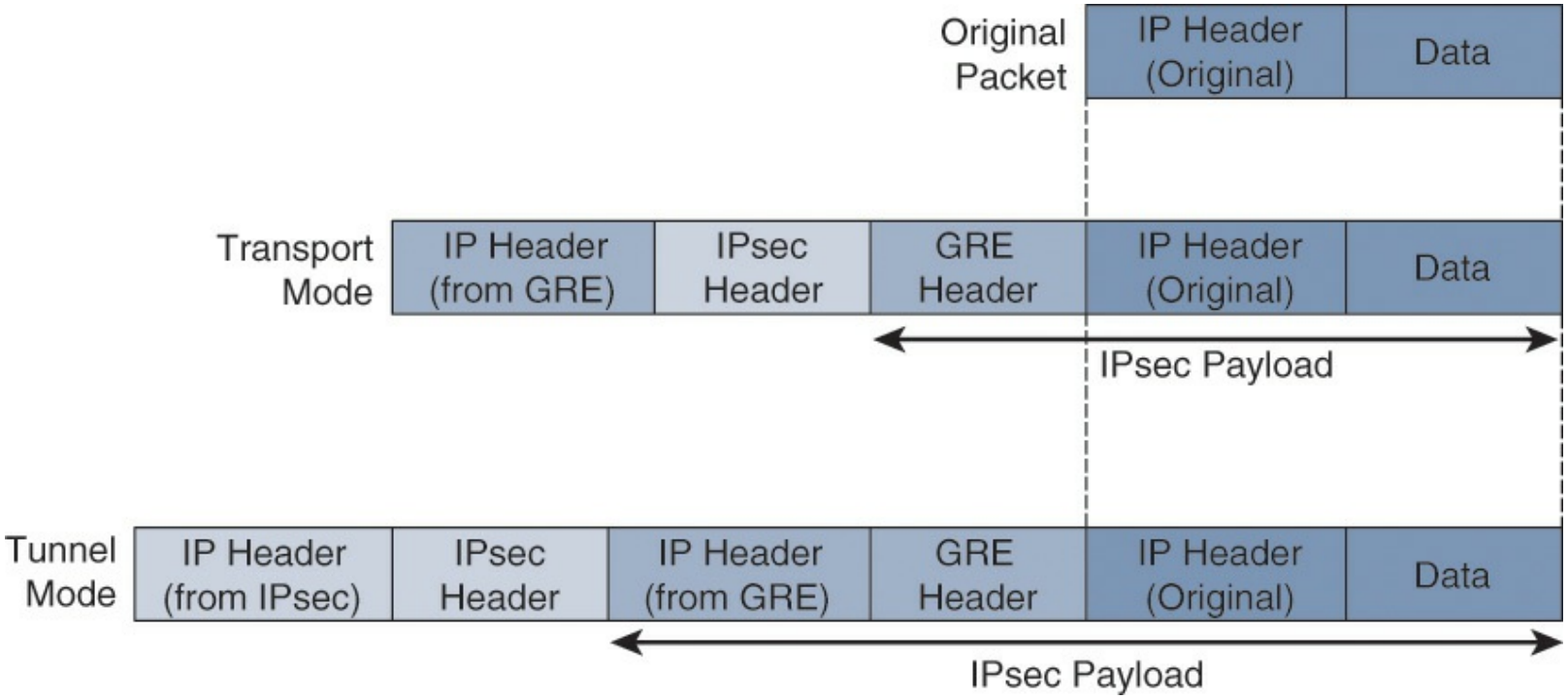


Figure 15-1 *GRE Overhead*

Table 15-2 summarizes the impact of IPsec mode, IPsec transport protocol (IPv4 or IPv6) and GRE mode (over IPv4 or IPv6) on the IP header overhead and the plaintext overhead.

IPsec Mode	GRE mode	IP Header Overhead	Plaintext Overhead
Transport	GRE/IPv4	20 bytes[1,4]	4 bytes[3]
	GRE/IPv6	40 bytes[1,4]	4 bytes[3]
Tunnel over IPv4	GRE/IPv4	20 bytes[2,4]	20 bytes[1,4] + 4 bytes[3]
	GRE/IPv6		40 bytes[1,4] + 4 bytes[3]
Tunnel over IPv6	GRE/IPv4	40 bytes[2,4]	20 bytes[1,4] + 4 bytes[3]
	GRE/IPv6		40 bytes[1,4] + 4 bytes[3]

[1]IP header added by GRE.

[2]IP header added by IPsec tunnel mode.

[3]Plus GRE options (if any).

[4]Plus IPv4 options or IPv6 non-IPsec extensions (if any).

Table 15-2 *GRE with IPsec Overhead*

Note

- In this chapter we consider only the case of GRE over IPsec. The opposite scenario, IPsec over GRE, is a rare but valid order of encapsulation that may be used in some specific designs. In that case, the IPsec overhead does not depend on the presence of GRE, since the GRE encapsulation takes place after IPsec encapsulation; thus the total overhead is the sum of the (outer) GRE overhead and the (inner) IPsec overhead.
- The IPsec and the GRE transport protocols (IPv4/IPv6) can differ when the IPsec and GRE tunnel endpoints are different. For example, IPsec tunnel mode of IPv4 with GRE/IPv6 or IPsec tunnel mode of IPv6 with GRE/IPv4.
- The IP header overhead refers to the overhead of the IP header in front of the IPsec header which depends only on the IP transport type (IPv4/IPv6) and is independent of IPsec transport or tunnel mode.
- The plaintext overhead refers to the extra bytes protected by IPsec in addition to the original IP packet.

Encapsulating Security Payload Overhead

Encapsulation Security Payload adds both a header and a trailer around the encapsulated packet. Figure 15-2 shows the general structure of an Encapsulation Security Payload packet (outer IP header not shown).

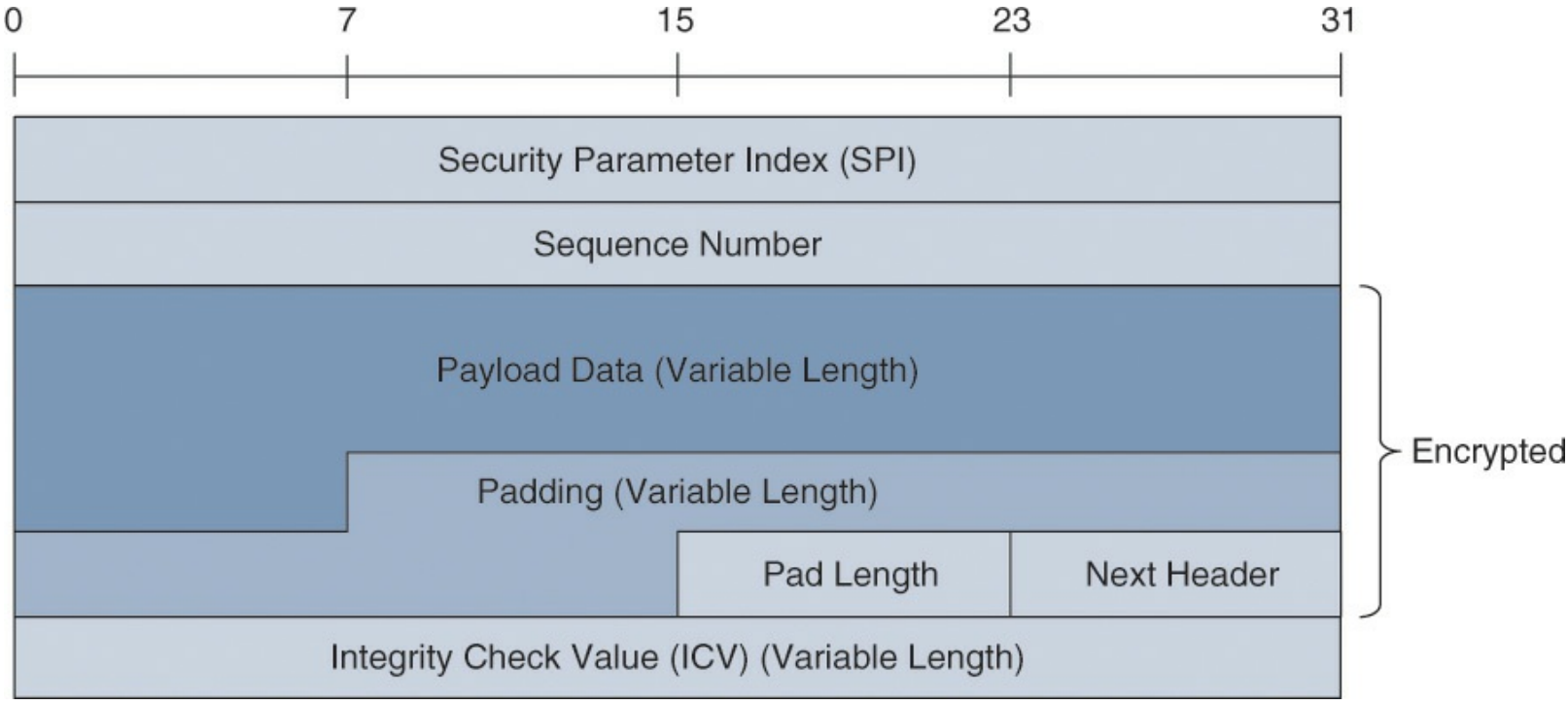


Figure 15-2 *Encapsulating Security Payload Format*

The Encapsulation Security Payload header contains two 32-bit (4-byte) values, the SPI and sequence number. The payload data that follows is the result of the encryption algorithm applied to the plaintext packet and may contain additional data on top of the ciphertext, for example an Initialization Vector (IV). The structure of the payload data is only relevant to the specific encryption algorithm that is used and is transparent to Encapsulation Security Payload.

The Encapsulation Security Payload trailer contains two 8-bit (1-byte) values the Pad Length, which

contains the number of padding bytes (see below), and the Next Header field, which contains the IP protocol number for the packet contained within the payload field. Those two fields are encrypted along with the payload data and are always the last 2 bytes of the encrypted portion of the Encapsulation Security Payload packet.

If an integrity algorithm is part of the transform set, the trailer also contains an ICV produced by the algorithm.

The variable-length padding (0 to 255 bytes) that precedes the trailer has two purposes:

- adjusting the plaintext (which includes the Pad Length and Next Header fields) to a multiple of the block size required by the block cipher used for encryption (if applicable)
- aligning the start of the ICV on a 32-bit (4-byte) boundary

The first item may not be required if the cipher operates as a stream cipher or if it allows for any plaintext input size without padding (like AES-GCM for example).

Based on this, the Encapsulation Security Payload overhead will be:

- 8 bytes for the header
- a certain amount of payload data (length depending on the encryption algorithm and the size of the plaintext packet, see "Encryption Overhead" later in this section)
- a certain amount of padding (length depending on the amount of payload data)
- 2 bytes for the fixed trailer fields
- a certain amount of ICV data (length depending on the integrity algorithm, see "Integrity Overhead" later in this section)

We will therefore need to examine the overhead for the various encryption and integrity algorithms before we can calculate the exact Encapsulation Security Payload overhead. Also note that the plaintext may include a GRE header and an IP header (see "GRE Overhead" earlier in this section).

When using a combined-mode algorithm (like AES-GCM), the ICV data may be accompanied by additional data such as the IV for the algorithm (see "Combined-mode Algorithm Overhead" later in this section).

Authentication Header Overhead

Authentication Header only adds a header in front of the encapsulated packet. Figure 15-3 shows the structure of an Authentication Header packet (outer IP header not shown).

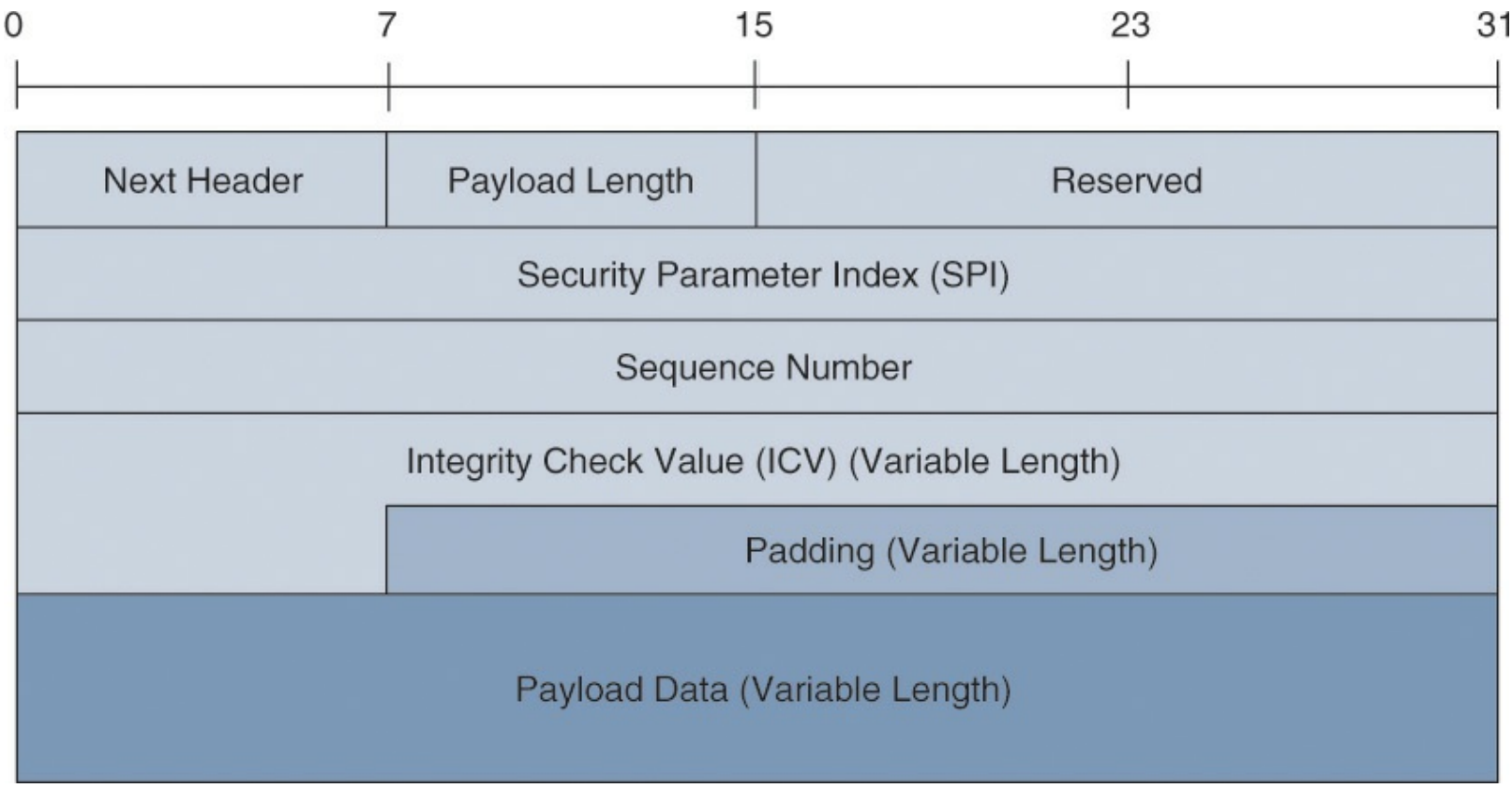


Figure 15-3 *Authentication Header Format*

The Authentication Header contains two 8-bit (1-byte) values: a Next Header field with the same meaning as in Encapsulation Security Payload and a Payload Length field that indicates the length of the Authentication Header (not the payload data length as the name seems to imply; see RFC4302 for details). The next 16 bits (2 bytes) are reserved and set to zero. Two 32-bit (4-byte) fields for SPI and sequence number follow. The total is thus 12 bytes.

The last part of the Authentication Header is the ICV produced by the integrity algorithm. If the ICV does not terminate on a 32-bit (4-byte, for IPv4 transport) or 64-bit (8-byte, for IPv6 transport) boundary, padding is added after the ICV so that the payload starts on the next boundary.

Based on this, the Authentication Header overhead will be

- 12 bytes for the fixed-length header fields
- a certain amount of ICV data (length depending on the integrity algorithm; see "Integrity Overhead" later in this section)
- 0 to 7 bytes of padding (depending on the ICV length and the transport protocol)

When a combined-mode algorithm (like AES-GMAC) is used with Authentication Header, the ICV data may be accompanied by additional data such as the IV for the algorithm. This additional data will have to be accounted for in the total overhead.

Encryption Overhead

The structure of the encrypted payload in Encapsulation Security Payload packets depends on the type and characteristics of the encryption algorithm. Block ciphers work on input and output blocks of a given length: before encryption, the input data must be padded to reach the block size, and after decryption, the padding must be removed to get back the original input data. This is referred to as explicit padding as it is actually carried as part of the encrypted payload, concealed within the ciphertext (Figure 15-4).

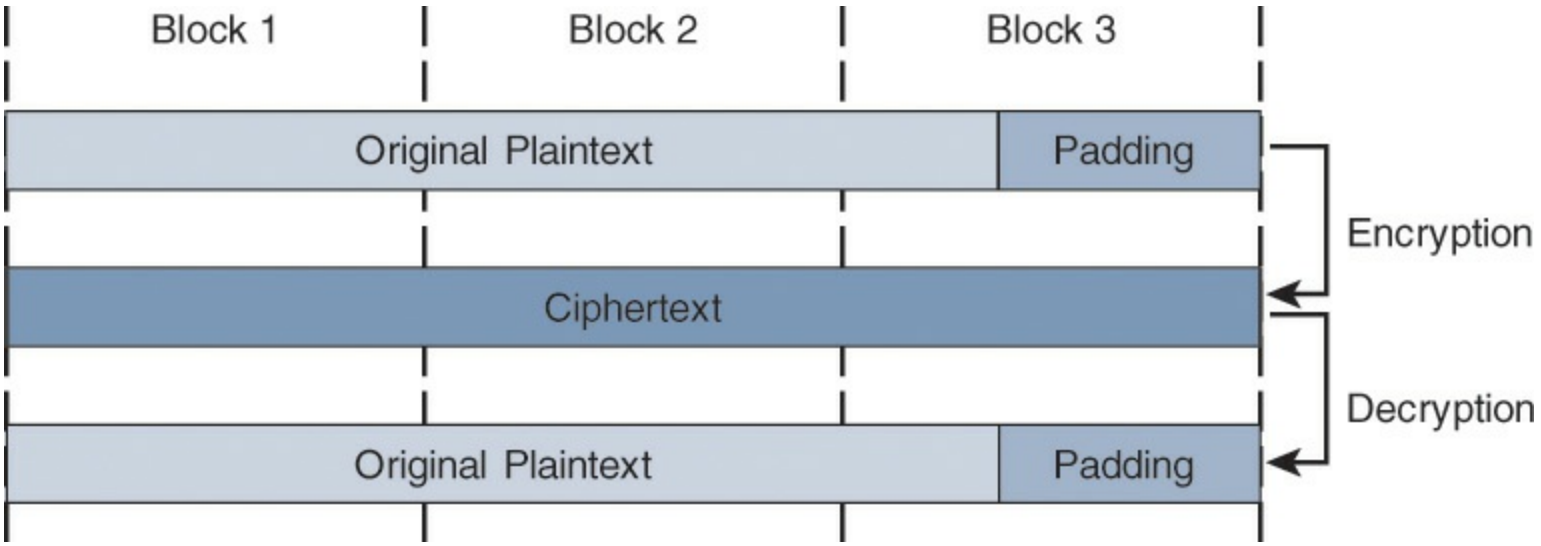


Figure 15-4 *Ciphertext Explicit Padding*

Algorithms that work in cipher block chaining (CBC) or Counter (CTR) mode require that an Initialization Vector be carried along with the encrypted payload. The IV typically comes first and is followed by the encrypted payload. The IV length is the same as the block size for the algorithm (note that the block size is not necessarily the same as the key size, for example AES-256 uses a 256-bit key and a 128-bit block).

Table 15-3 lists the block/IV and key size for the most common encryption algorithms, as well as the most recent RFC that defines their use in IPsec. The NULL algorithm (Encapsulation Security Payload without encryption) produces output that is the same as its input, thus no input padding is required.

	Block and IV Size	Key Size	RFC
NULL	N/A	N/A	2410
3DES-CBC[1]	64 bits (8 bytes)	168 bits	2451
AES-CBC	128 bits (16 bytes)	128, 192, or 256 bits	3602
AES-CTR	128 bits (16 bytes)	128, 192, or 256 bits	3686

[1]Also known as DES-EDE3-CBC.

Table 15-3 *Algorithm Block, IV, and Key Sizes*

Integrity Overhead

Many integrity algorithms are based on some sort of one-way-hash function; a few others use an encryption algorithm (for example, AES-XCBC-MAC-96 is based on 128-bit AES). Different hash functions or algorithms have different output sizes, and the resulting hash is sometimes truncated to a certain length to produce the ICV. For example, the outputs of SHA-1 and MD5 are 160 bits (20 bytes) and 128 bits (16 bytes) long, respectively, but they are all truncated to 96 bits (12 bytes) for use in IPsec.

Most origin authentication and integrity algorithms use HMAC, an algorithm defined in RFC2104 that provides message authentication based on a given hash function and a secret key. The output size of HMAC is the same as that of the underlying hash function, but it may also be truncated to produce a shorter ICV.

For the protection of IPsec Security Associations, the secret key is derived from the attributes

negotiated in the IKE_SA_INIT exchange;

KEYMAT = prf + (SK_d, Ni | Nr)

Where SK_d is generated from the IKE_SA_INIT exchange.

Additionally at rekey when new IPsec Security Associations are created using the formula;

KEYMAT = prf + (SK_d, g^{ir} (new) | Ni | Nr)

Where g^{ir} (new) is the shared secret from the ephemeral Diffie-Hellman exchange.

Table 15-4 lists the most common integrity algorithms, the output size of the hash function they are based on (if applicable), the actual length of the produced ICV, and the most recent RFC that defines their use in IPsec.

	Base Output Size	ICV Size	RFC
HMAC-MD5-96	128 bits (16 bytes)	96 bits (12 bytes)	2403
HMAC-SHA-1-96	160 bits (20 bytes)	96 bits (12 bytes)	2404
AES-XCBC-MAC-96	128 bits (16 bytes)	96 bits (12 bytes)	3566
AES-CMAC-96	128 bits (16 bytes)	96 bits (12 bytes)	4494
AES-GMAC[1]	128 bits (16 bytes)	192 bits (24 bytes) (incl. 64-bit IV)	4543
HMAC-SHA-256-128[2]	256 bits (32 bytes)	128 bits (16 bytes)	4868
HMAC-SHA-384-192[2]	384 bits (48 bytes)	192 bits (24 bytes)	4868
HMAC-SHA-512-256[2]	512 bits (64 bytes)	256 bits (32 bytes)	4868

[1]Authentication Header-only integrity variant (see "Combined-mode Algorithm Overhead").

[2]SHA-256/384/512 are part of the SHA-2 family.

Table 15-4 *HMAC Base and ICV Output*

These integrity algorithms have different input block sizes, and the input data may require padding when computing the ICV. However this padding is only used during computation and is never sent as part of the packet (it is referred to as implicit padding); therefore Table 15-4 makes no mention of the input block size since it has no bearing on the integrity overhead in the output packet.

Combined-mode Algorithm Overhead

Since IPsec-v3, encryption and integrity protection with Encapsulation Security Payload can be achieved using a single algorithm called a combined-mode algorithm (which returns both a ciphertext and an ICV), like AES-GCM (Galois Counter Mode) for example.

A particular case of such an algorithm is AES-GMAC, which only provides integrity but is available both as an integrity algorithm (for use in Authentication Header) and a combined-mode algorithm (for use in Encapsulation Security Payload). Both variants require the transmission of an IV; this sets the integrity variant of GMAC aside from other integrity algorithms that only add an ICV to the packet.

Table 15-5 shows the most common combined-mode algorithms, which share a number of characteristics:

- any input size is allowed (no input padding is required)
- the output ciphertext is the same length as the input plaintext
- an IV needs to be added to the Encapsulation Security Payload packet
- AES is the underlying algorithm and the three AES key sizes are supported

	IV Size	ICV Size	Key Size	RFC
AES-GCM	64 bits (8 bytes)	128 bits (16 bytes)[1]	128, 192 or 256 bits	4106
AES-CCM	64 bits (8 bytes)	128 bits (16 bytes)	128, 192 or 256 bits	4309
AES-GMAC[2]	64 bits (8 bytes)	128 bits (16 bytes)	128, 192 or 256 bits	4543

[1]ICV of 8 and/or 12 bytes may also be supported depending on the implementation.

[2]Encapsulation Security Payload-only combined variant, without encryption.

Table 15-5 *Combined-mode-Algorithm IV, ICV, and Key Sizes*

Plaintext MTU

The IPsec overhead does not depend only on the transform set; it also varies with the length of the packet due to the presence of padding. Table 15-6 shows the encrypted packet length and total overhead for selected plaintext packet lengths when using Encapsulation Security Payload tunnel mode, AES-CBC 128 and HMAC-SHA-1-96. The overhead varies between 58 and 73 bytes.

Plaintext IP Packet Length	Encrypted Packet Length	IPsec Overhead
1421 bytes	1480 bytes	59 bytes
1422 bytes	1480 bytes	58 bytes[1]
1423 bytes	**1496 bytes**	73 bytes[2]
...	...	...
1430 bytes	1496 bytes	66 bytes
1431 bytes	1496 bytes	65 bytes
...	...	...
1438 bytes[3]	1496 bytes	58 bytes[1]
1439 bytes	**1512 bytes**	73 bytes[2]
1440 bytes	1512 bytes	72 bytes
...	...	...

[1]Smallest overhead.

[2]Largest overhead.

[3]Largest possible plaintext length for an MTU between 1496 and 1511 (included).

Table 15-6 *Varying Overhead for AES-CBC 128 and HMAC-SHA-1-96*

Internally, Cisco IOS computes the IPsec overhead by considering the IP MTU on the output interface and finding the largest possible plaintext packet that will fit inside that MTU after encapsulation.

Table 15-6 tells us that for an MTU of 1500 bytes, the largest possible plaintext packet is 1438 bytes; therefore the overhead as computed by IOS for these transforms and for an MTU of 1500 is 62 bytes.

This is the best way to compute the highest possible plaintext MTU: any plaintext packet with a length of (1500 – 62) = 1438 bytes or smaller will not require fragmentation. Note that this value is not the true IPsec overhead for the packet, which is (1496 – 1438) = 58 bytes. The disadvantage of this method is that it requires a new computation every time the output IP MTU changes.

Maximum Overhead

Another, simpler method to find a safe value for the plaintext MTU is to find an upper bound for the overhead. This method has the advantage of returning a worst case value that can be trusted regardless of the output interface MTU, at the expense of a slightly suboptimal plaintext MTU.

Table 15-7 shows the overhead characteristics of the encryption, combined-mode and integrity algorithms that were mentioned so far, summarized into the following values:

- **Maximum input padding:** the largest amount of input data padding required for the plaintext to reach the appropriate input size. For integrity algorithms the value is N/A as implicit padding is used (see "Integrity overhead").
- **Maximum output overhead:** the largest amount of output data produced by the algorithm. It can be an Initialization Vector, an integrity check value or both. For encryption and combined-mode algorithms, this comes on top of the ciphertext (which is the same length as the original plaintext plus the input padding).
- **Maximum ICV padding:** for encryption algorithms, the largest amount of padding required before the ICV (if present) to align it with a 4-byte boundary; for integrity algorithms (only when used in Authentication Header), the largest amount of padding required after the ICV to align the payload to a 4- or 8-byte boundary (see "Authentication Header Overhead").

	Type	Max. Input Padding	Max. Output Overhead	Max. ICV Padding
NULL	Encryption	0 bytes	N/A	3 bytes
3DES-CBC	Encryption	7 bytes	8 bytes (IV)	0 bytes
AES-CBC	Encryption	15 bytes	16 bytes (IV)	0 bytes
AES-CTR	Encryption	15 bytes	16 bytes (IV)	0 bytes
AES-GCM	Combined	0 bytes	24 bytes (IV + ICV)	3 bytes
AES-CCM	Combined	0 bytes	24 bytes (IV + ICV)	3 bytes
AES-GMAC[1]	Combined, Integrity	0 bytes	24 bytes (IV + ICV)	3 bytes
HMAC-MD5-96	Integrity	N/A	12 bytes (ICV)	0 bytes (IPv4)[2] 4 bytes (IPv6)[2]
HMAC-SHA-1-96	Integrity	N/A	12 bytes (ICV)	0 bytes (IPv4)[2] 4 bytes (IPv6)[2]
AES-XCBC-MAC-96	Integrity	N/A	12 bytes (ICV)	0 bytes (IPv4)[2] 4 bytes (IPv6)[2]
AES-CMAC-96	Integrity	N/A	12 bytes (ICV)	0 bytes (IPv4)[2] 4 bytes (IPv6)[2]
HMAC-SHA-256-128	Integrity	N/A	16 bytes (ICV)	0 bytes
HMAC-SHA-384-192	Integrity	N/A	24 bytes (ICV)	0 bytes
HMAC-SHA-512-256	Integrity	N/A	32 bytes (ICV)	0 bytes

[1]Both variants (end result is always 8 bytes IV + 16 bytes ICV).

[2]Padding between ICV and authenticated data (only when used in Authentication Header).

Table 15-7 *Worst Case Maximum Overhead*

In this list, all encryption algorithms that require the input to be aligned to a certain block size (that is, all except NULL, AES-GCM, AES-CCM, and AES-GMAC) have a block size that is a multiple of 4 bytes. Therefore, after encryption the ICV (if present) is already aligned on a 4-byte boundary as required by Encapsulation Security Payload, and no extra padding is necessary. For the others, when they are used in combination with data integrity (which is the case by definition for the three combined-mode algorithms and is mandated by the RFC for ESP-NULL), up to 3 additional bytes may be required to align the ICV.

Maximum Encapsulation Security Payload Overhead

The maximum Encapsulation Security Payload overhead for a given encryption or combined-mode algorithm and an optional integrity algorithm is the sum of:

- 20 bytes (tunnel mode over IPv4 only)
- 40 bytes (tunnel mode over IPv6 only)
- 24 bytes (GRE/IPv4 only) + 4 bytes (if tunnel key configured)
- 44 bytes (GRE/IPv6 only) + 4 bytes (if tunnel key configured)
- 10 bytes (fixed header/trailer fields)
- the maximum input padding of the encryption/combined-mode algorithm
- the maximum output overhead of the encryption/combined-mode algorithm
- the maximum ICV padding of the encryption/combined-mode algorithm
- the maximum output overhead of the integrity algorithm (if configured)

Maximum Authentication Header Overhead

The maximum Authentication Header overhead for a given integrity algorithm is the sum of:

- 20 bytes (tunnel mode over IPv4 only)
- 40 bytes (tunnel mode over IPv6 only)
- 24 bytes (GRE/IPv4 only) + 4 bytes (if tunnel key configured)
- 44 bytes (GRE/IPv6 only) + 4 bytes (if tunnel key configured)
- 12 bytes (fixed header fields)
- the max. output overhead of the integrity algorithm
- the max. ICV padding of the integrity algorithm

Extra Overhead

The above formulas do not account for additional overhead that may exist due to additional encapsulation of specific features, such as IPv4 options or IPv6 extension headers (for example, routing headers) and UDP encapsulation for NAT traversal that may be added to the IPsec packet.

Example 1

Consider IPv4 Encapsulation Security Payload tunnel mode with AES-CBC-128 and HMAC-SHA-1-96. The contributing factors to the IPsec overhead are:

- 20 bytes for the tunnel mode IPv4 header
- 10 bytes for the fixed header/trailer fields
- up to 15 bytes for the encryption padding
- 16 bytes for the encryption overhead (IV)
- 12 bytes the for integrity overhead (ICV)

This produces a total of 73 bytes, which matches the highest number in Table 15-6.

The smallest plaintext IPv4 packet size that would result in this worst case overhead is 31 bytes (20 bytes IPv4 header + 11 bytes payload). After adding the Pad Length and Next Header fields (1 byte each), the extended plaintext is 33 bytes long and requires 15 bytes of padding to reach 48 bytes and be a multiple of the 16-byte AES block size.

Figure 15-5 shows the structure of the packet after encapsulation.

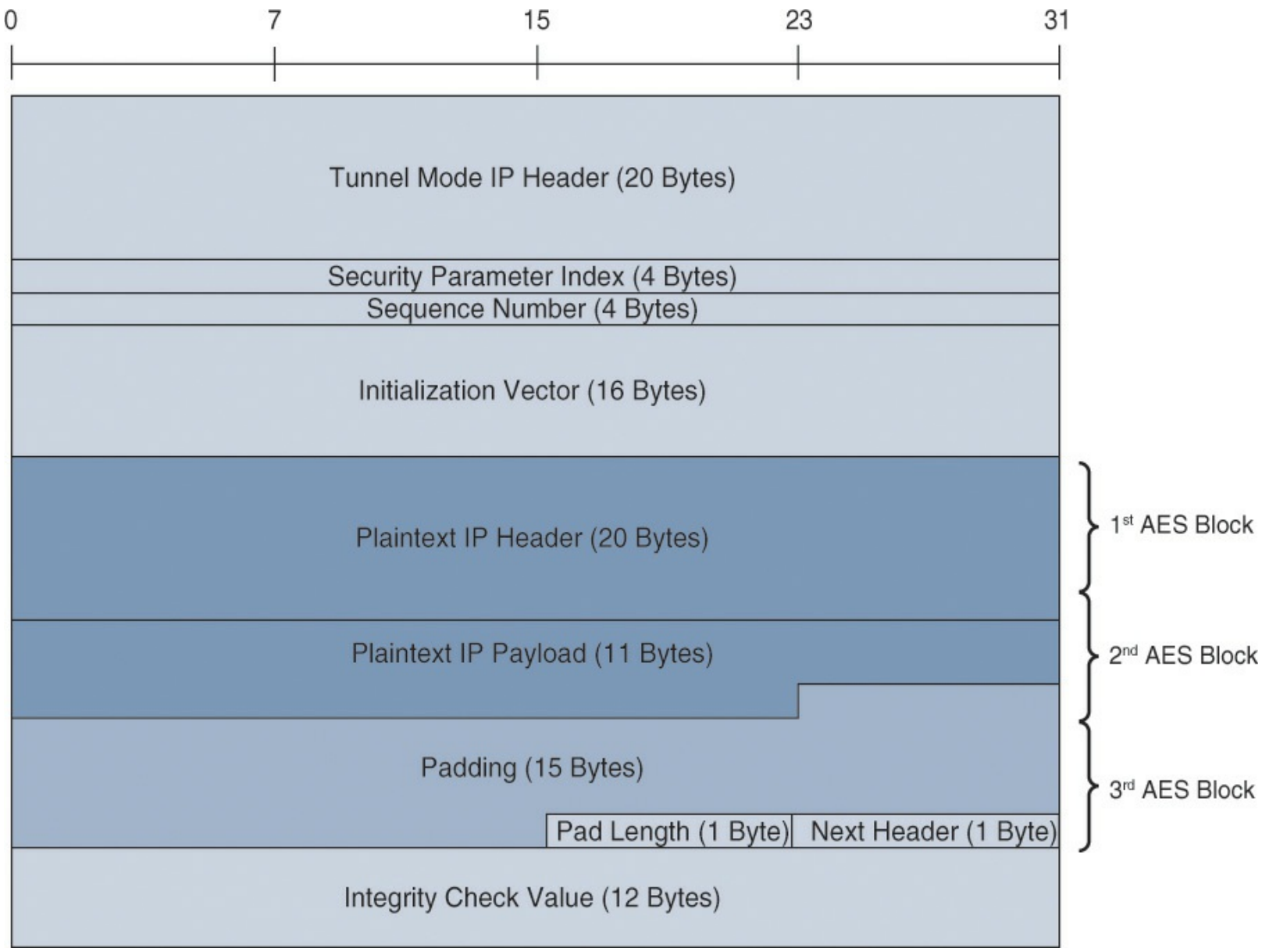


Figure 15-5 *Packet Structure Post-Encapsulation*

Example 2

Consider GRE/IPsec over IPv6 with Encapsulation Security Payload transport mode and AES-GCM-256. A GRE tunnel key is configured. The contributing factors to the IPsec overhead are:

- 48 bytes for GRE (40 bytes for IPv6, 4 bytes for GRE, 4 bytes for the tunnel key)
- 10 bytes for the fixed header/trailer fields
- 24 bytes for the combined-mode encryption and integrity overhead (IV and ICV)
- up to 3 bytes for the ICV padding

This produces a total of 85 bytes.

Please refer the Cisco IPsec Overhead Calculator tool at https://cway.cisco.com/tools/ipsec-overhead-calc/ipsec-overhead-calc.html for a quick calculation of IPsec overhead based on various input parameters such as plaintext packet size, IPsec mode, protocols, and algorithms.

IPsec and Fragmentation

This section describes the factors that determine the fragmentation of IP packets before and after IPsec encryption and the impact of fragmentation and reassembly.

Maximum Transmission Unit

The maximum transmission unit (MTU) of a link is the largest protocol data unit (PDU) that can be carried by a Layer 2 link. On a regular Ethernet link the MTU is 1500 bytes, meaning that an Ethernet frame will carry up to 1500 bytes of payload data. The frame as seen on the wire is actually larger, since Ethernet adds a 14-byte Ethernet header (6 bytes for the destination MAC address, 6 bytes for the source MAC address and 2 bytes for the EtherType value), an optional 802.1Q VLAN tag (4 bytes) and a frame check sequence (4 bytes) for a total of up to 1518 bytes.

Additional layers of encapsulation may reduce the MTU of the link; for example, PPP over Ethernet (PPPoE) adds 8 bytes of overhead for a resulting MTU of (1500 − 8) = 1492 bytes.

On a Cisco router, the MTU of a physical interface is set by the interface driver. Some interface types (for example, virtual Ethernet interfaces) have a default MTU that can be overridden using the **mtu** command, to make room for additional encapsulation or adjust the MTU to that of the underlying physical link. The interface MTU takes effect for all protocols and is displayed in the output of **show interfaces** (as shown in the following example).

Click here to view code image

```
Router#show interfaces FastEthernet4
FastEthernet4 is up, line protocol is up
  Hardware is PQUICC_FEC, address is 0021.5528.7f84 (bia 0021.5528.7f84)
  Internet address is 10.48.66.98/23
  MTU 1500 bytes, BW 100000 Kbit/sec, DLY 100 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation 802.1Q Virtual LAN, Vlan ID  1., loopback not set
  Keepalive set (10 sec)
```

Separate MTU values specific to the IP protocol family can be configured using the **ip mtu** and **ipv6 mtu** commands; those MTU values do not affect any protocols other than IPv4 and IPv6. The IP MTU is displayed in the output of **show ip interfaces** and **show ipv6 interfaces** (as shown in the following example).

Click here to view code image

```
Router#show running-config interface FastEthernet4
interface FastEthernet4
 ip address 10.0.0.1 255.255.255.0
 ip mtu 1400
 ipv6 enable
 ipv6 mtu 1300
end

Router#show ip interface FastEthernet4
FastEthernet4 is up, line protocol is up
  Internet address is 10.0.0.1/24
  Broadcast address is 255.255.255.255
  Address determined by setup command
  MTU is 1400 bytes
  ...

Router#show ipv6 interface FastEthernet4
FastEthernet4 is up, line protocol is up
  IPv6 is enabled, link-local address is FE80::221:55FF:FE28:7F84
  No Virtual link-local address(es):
  No global unicast address is configured
```

```
  Joined group address(es):
    FF02::1
    FF02::1:FF28:7F84
  MTU is 1300 bytes
  ...
```

The length of the forwarded packets is checked against the interface MTU before the interface encapsulation in the outbound direction only. If the packet is larger than the MTU (or the relevant IP MTU), it cannot be forwarded as-is. In the case of IPv4 traffic, it must be fragmented (if allowed) before being forwarded. In the case of IPv6 traffic, the packet must be dropped, and the originating host must be notified.

Fragmentation in IPv4

IPv4 packets may be fragmented at any point along the path from source to destination. In addition, fragments of a packet may be further fragmented along the path to the destination. Reassembly is always performed by the destination host. This does not mean that intermediate nodes may not process the fragments (for example, by firewalls or IPS performing virtual reassembly, buffering the fragments in order to perform inspection of the reassembled packet), but the packet remains fragmented on the wire.

Fragmentation involves three fields in the IPv4 header (illustrated in Figure 15-6):

- the 16-bit (2-byte) Identification field (sometimes referred to as the IP ID), an identifier for fragments of an original packet
- the 3-bit Flags field, containing the Do not Fragment (DF) and More Fragments (MF) flags (1 bit each + 1 reserved)
- the 13-bit Fragment Offset field, indicating the offset (in 8-byte units) of the current fragment within the original packet

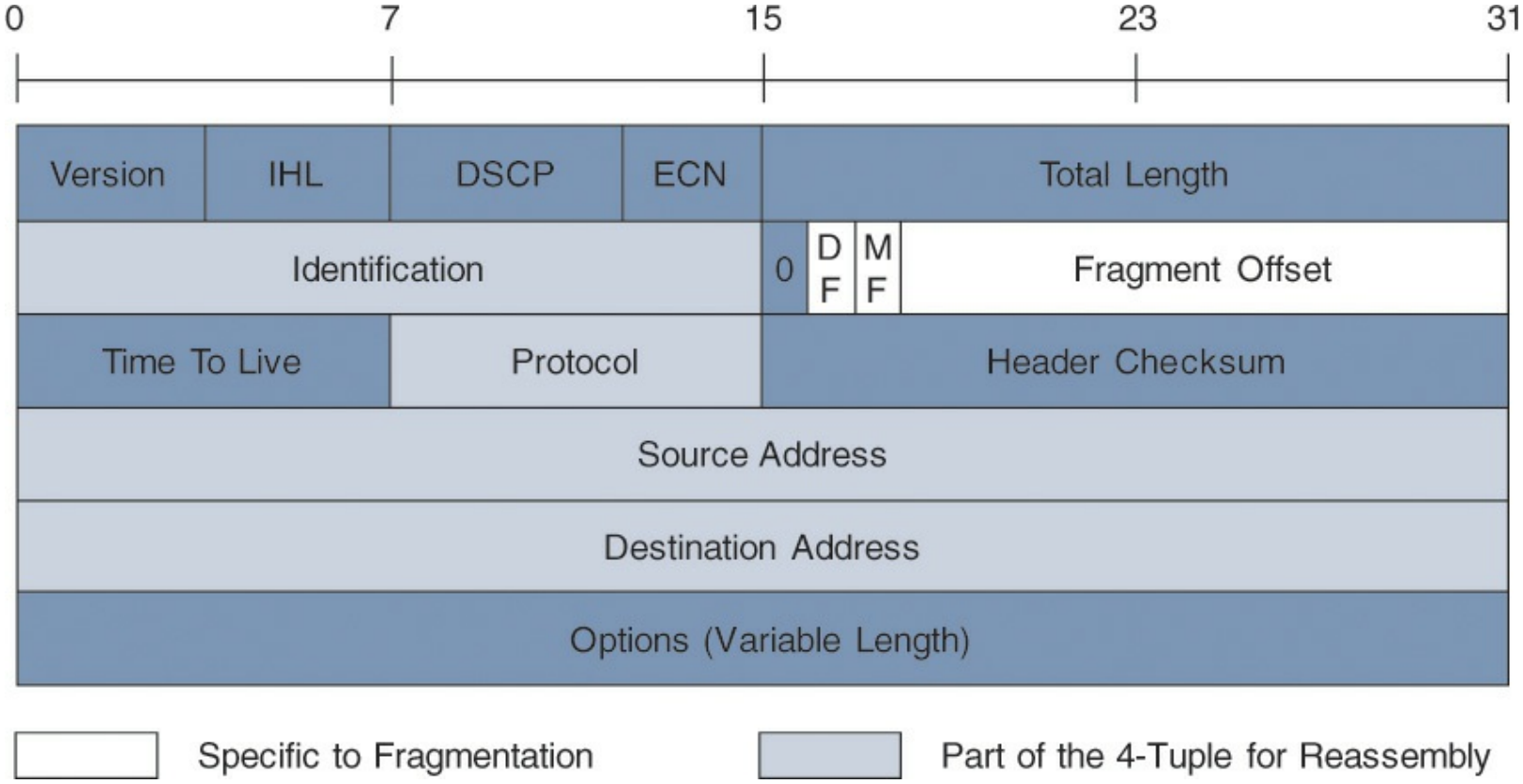


Figure 15-6 *IP Header and Fragmentation*

RFC791 specifies that the Identification value of an IPv4 packet must be different from that of all other IPv4 packets from the same flow (source/destination/protocol 3-tuple) for the time the packet will be present on the network. In practice with today's high-speed networks, a 2-byte field has become much too small for such a requirement; this may lead to problems with reassembly on high-throughput links (see "The Impact of Fragmentation"). The IPv6 protocol avoids such issues by taking a completely different approach towards fragmentation (see "Fragmentation in IPv6").

Upon creation of an IP datagram, the MF bit and Fragment Offset field are set to 0. The DF bit may be set to 0 or 1, depending on the local configuration and PMTUD settings (see "Path MTU Discovery"). If at any point on the path (this can be on the originating node or on any intermediate node) the packet is too large for the outgoing interface MTU, the forwarding node will react based on the value of the DF bit in the IP header.

If the DF bit is set to 1, the packet will be dropped, and an ICMP Destination Unreachable packet (type 3, code 4: fragmentation needed and DF set) will be sent back to the source of the dropped packet, including the MTU value that triggered the drop so that the source host can adjust its packet size (see "Path MTU Discovery") as well as a portion of the packet (IP header + first 64 bits of payload). The ICMP notification is generated only if the interface that the packet was received on has the **ip unreachables** command configured, which is the default configuration.

If the DF bit is set to 0, the IP payload must be split across two or more new IP packets. The original IP header is copied into the header portion of the new fragment packets, and the header fields are adjusted as follows:

- the MF field is set to 1 in all but the last fragment
- the Fragment Offset reflects the offset of the payload in each fragment within the payload of the original unfragmented packet, in units of 8 bytes, starting at 0 in the first fragment
- the Total Length and Header Checksum are adjusted for each fragment

Figure 15-7 shows a schematic view (not to scale) of IPv4 fragmentation for a 1500-byte UDP packet going through an interface with a 1400-byte MTU. The IP payload is composed of an 8-byte UDP header and a 1472-byte UDP payload. The first fragment contains the UDP header and the beginning of the UDP payload, while the second fragment only contains the end of the UDP payload. Note that for clarity the Fragment Offset (FO) for the second fragment is set to 1400; however, within the IP header this is value is defined as a multiple of 8 octets, so it would actually be seen as 1400/8 = 175.

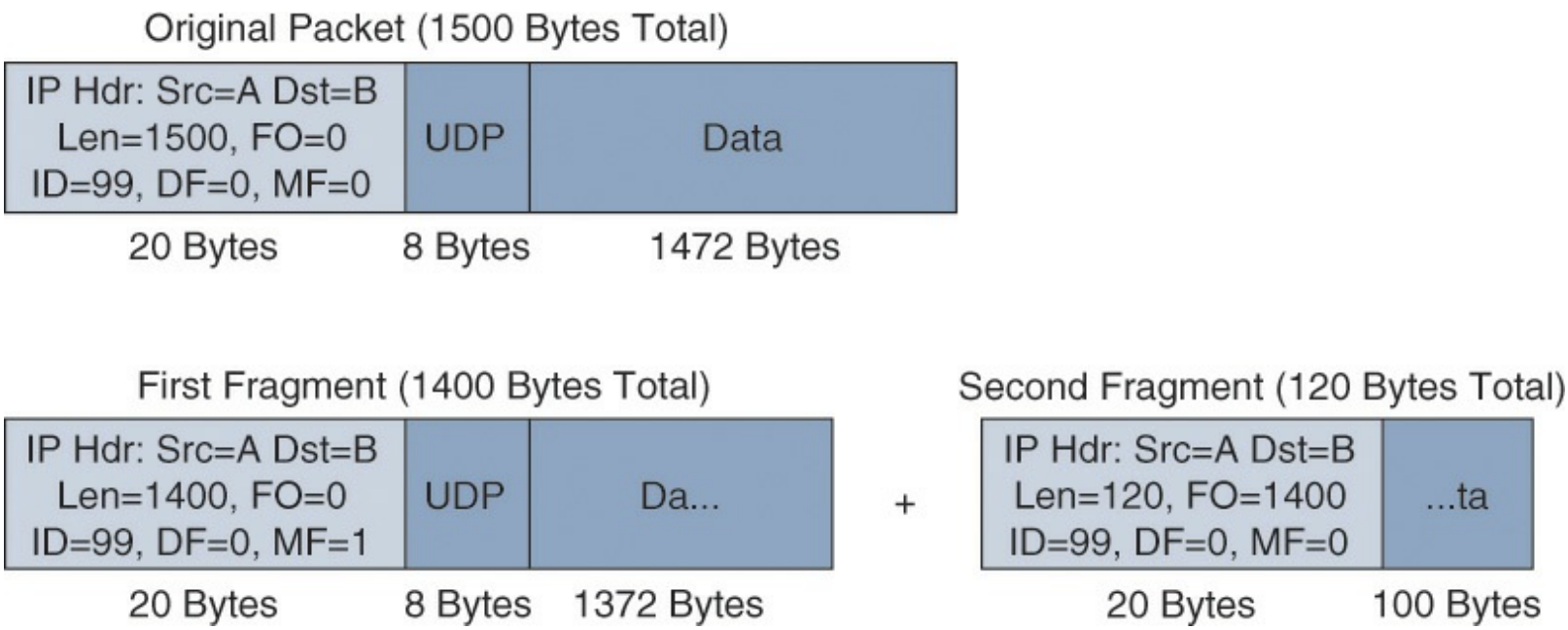


Figure 15-7 *IPv4 Fragmentation*

The packet may be split in multiple ways depending on the software or hardware performing the fragmentation. A 1500-byte payload going through an interface with a 1400-byte MTU could be split as:

- a first fragment of 1400 bytes and a second fragment of 120 bytes
- a first fragment of 120 bytes and a second fragment of 1400 bytes
- a first fragment of 752 bytes and a second fragment of 748 bytes

The only constraint is that the payload length of all but the last fragment must be a multiple of 8 bytes (because of the Fragment Offset field).

Reassembly is performed only on the destination host, by buffering fragments as they arrive until the complete packet can be reassembled. Fragments of a single original packet are identified based on the source, destination, protocol, identification 4-tuple. If not all fragments are received within a certain amount of time, the already received ones are discarded to free up memory. There are no retransmissions; thus the loss of a single fragment means the loss of the entire packet.

The **show ip traffic** command on IOS and **show platform hardware qfp active feature ipfrag global** on IOS-XE allows for viewing of traffic statistics pertaining to fragmentation of IPv4 traffic. The following example illustrates the use of this command.

Click here to view code image

```
Router#show ip traffic | include reassembled|fragmented
  Frags: 7 reassembled, 0 timeouts, 0 couldn't reassemble
         7 fragmented, 14 fragments, 0 couldn't fragment
```

Fragmentation in IPv6

IPv6 takes a different approach than IPv4 when it comes to fragmentation. In IPv6, packets are never fragmented on the path, only on the originating host. A packet that is too large for the egress MTU must be dropped, and an ICMPv6 message (type 2, code 0: packet too big) must be generated, similar to the way IPv4 behaves when the DF bit is set. The ICMPv6 notification is generated only if the interface that the packet was received on (typically the one facing the source host) has **ipv6 unreachables** configured, which is the default.

This behavior enables the source host to perform Path MTU Discovery (see next topic) and adapt its Path MTU to send smaller packets. Any Layer 2 link used to carry IPv6 must be capable of forwarding datagrams of up to 1280 bytes (IPv6 header included).

Fragmentation is signaled through the addition of a Fragment extension header in the IPv6 fragments. An IPv6 packet that needs to be fragmented is first split into two parts:

- an unfragmentable part: the IPv6 header plus some of the extension headers, if any (those that must be processed by nodes on the way to the destination)
- a fragmentable part: the rest of the extension headers, if any, and the IPv6 payload data

First the unfragmentable part is replicated into all fragments, and a Fragment extension header is added at the end. Figure 15-8 shows the structure of the IPv6 Fragment extension. It contains three fields with similar semantics as their counterparts in IPv4:

- the 13-bit Fragment Offset field is the offset in 8-byte units of the fragment payload in the original packet payload (nonzero in all fragments except the first)
- the M flag has the same meaning as the MF flag in IPv4 (more fragments follow; it is set to 1 in all fragments except the last)
- the 32-bit Identification field used to correlate fragments from the same original packet

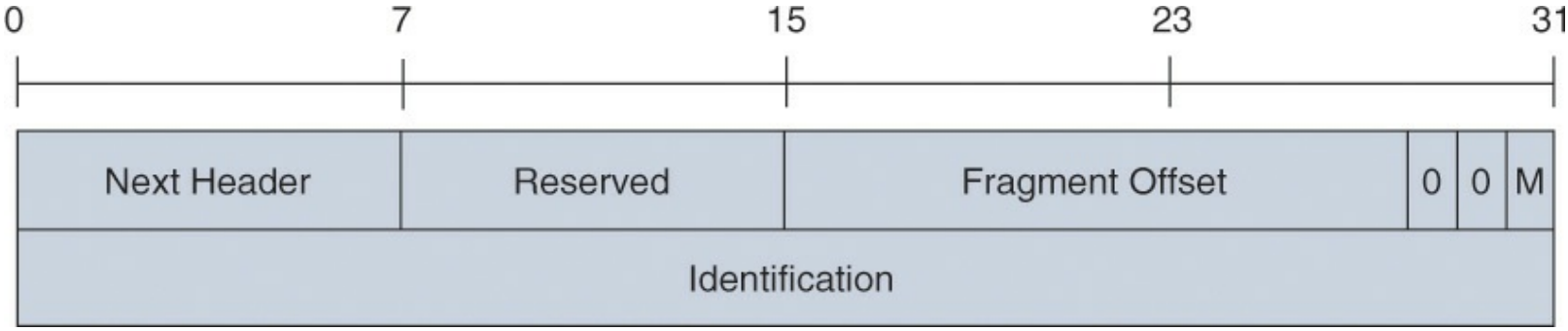


Figure 15-8 *IPv6 Fragmentation Extension*

An equivalent to the IPv4 DF bit is not required, since it is forbidden to fragment IPv6 packets in transit. The Identification field is 32 bits long and populated only when the Fragment extension is present, which makes collisions virtually impossible.

Reassembly is performed by the destination host based on the source/destination/protocol/identification 4-tuple. All fragments must be received within 60 seconds starting with the reception of the first-arriving fragment (not necessarily that with an offset of zero, as fragments may be reordered in transit); after that time, if one or more fragments are missing, the ones that have been received are discarded. If the first fragment (with an offset of zero) was received, an ICMPv6 Time Exceeded message (type 3, code 1: fragment reassembly time exceeded) is sent back to the source.

The **show ipv6 traffic** command on IOS and **show platform hardware qfp active feature ipfrag global** on IOS-XE allows for viewing of traffic statistics pertaining to fragmentation of IPv6 traffic. The following example illustrates the use of this command.

Click here to view code image

```
Router#show ipv6 traffic | include reassembled|fragmented
            17 fragments, 8 total reassembled
            10 fragmented into 20 fragments, 0 failed
```

Path MTU Discovery

By default, a source host that is sending IP traffic out of an interface will send IP datagrams that are as big as the MTU on the output interface will allow. For a standard Ethernet interface without jumbo frame support, IP datagrams will be 1500 bytes long (IP header included, that is, 1480 bytes of payload if no IP options are present).

This packet size may not be optimal, as the MTU one or more hops away may be lower than 1500 due to different link types or encapsulation, requiring fragmentation on the path and reassembly at the destination host. RFC1191 describes a technique known as Path MTU Discovery (PMTUD) for dynamically learning about the lowest MTU on the path to a certain IPv4 destination and adjusting the size of output packets accordingly. Some of the considerations in RFC1191 are outdated and no longer apply to modern networks; however the basic principles it outlines are still in use today. PMTUD for IPv6 is defined in RFC1981 and is based on the same principles.

In IPv4, PMTUD requires setting the DF bit in the IP header of all generated packets. In IPv6, the DF bit does not exist and is implicit: IPv6 traffic may not be fragmented except by the source host itself. In both cases, if the packet happens to be too large to be forwarded by one of the hops on the path, it is discarded, and an ICMP or ICMPv6 message of the appropriate type (often simply referred to as Cannot Fragment or CF) is generated and sent back to the source indicating the MTU value that caused the drop.

A source host with PMTUD enabled will process this notification, update a per-destination or per-path record, and inform higher-layer protocols (for example, TCP) of the change. The intent is to keep the higher layer (referred to as the packetization layer) informed of the maximum size of the data that it may send to IP for forwarding. Most modern operating systems perform PMTUD by default.

Figure 15-9 shows the basic principle of Path MTU Discovery with IPv4.

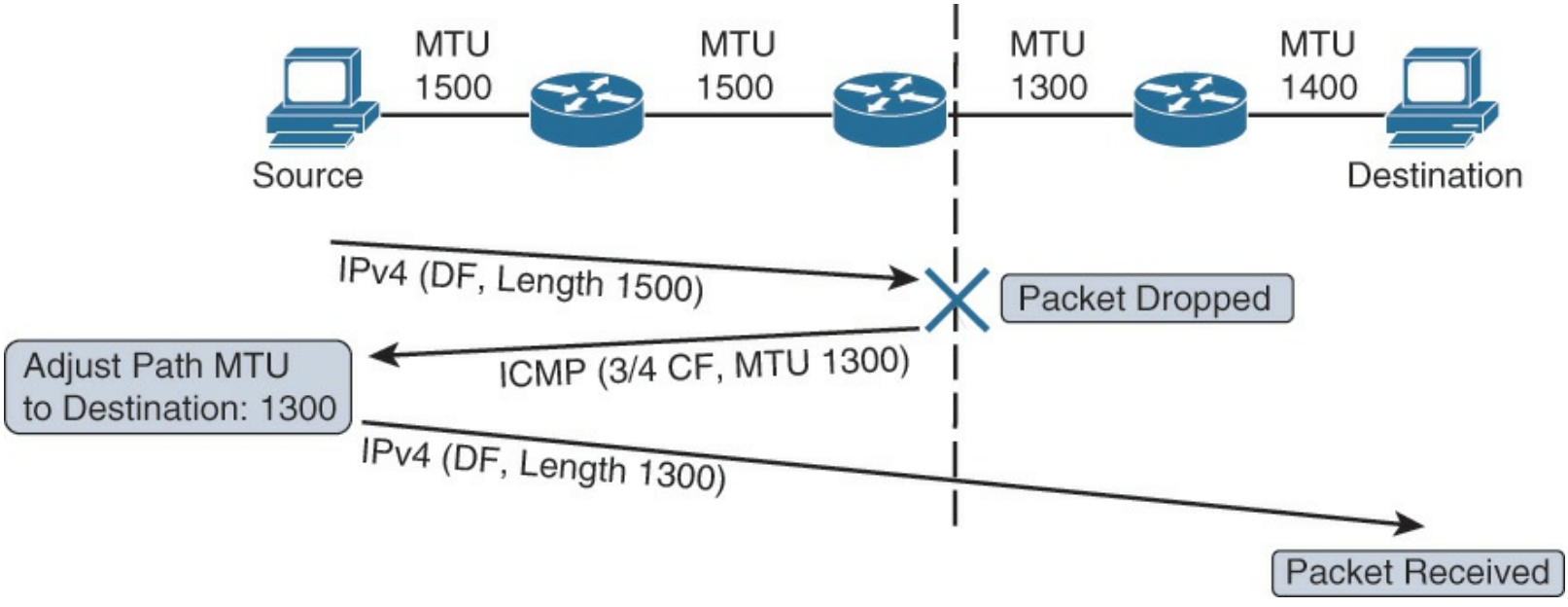


Figure 15-9 *Path MTU Discovery*

This mechanism remains enabled at all times, so that an MTU change in the middle of a connection can be detected. While MTU discovery generally occurs at the beginning of a data transfer, it can take place multiple times afterwards depending on the evolution of packet sizes received from the upper layers and/or changes on the path to the destination.

There is a common misconception that PMTUD is only supported in conjunction with TCP. That is the

case only for certain implementations; PMTUD can theoretically work with any protocol that can adapt its datagram size (for an example, see "Tunnel PMTUD" later in this chapter). It is true however that a connection-based protocol like TCP is naturally well-adapted to keeping track of the maximum allowed packet size, since it already maintains descriptor structures on a per-connection basis.

Path MTU Discovery is based on negative signaling. No news is good news in theory, but not always in practice: the absence of signaling may be due to ICMP filtering on the return path towards the source. Specific precautions may be required if this is the case. Also, there is no positive signaling if the MTU goes up (typically when the traffic gets rerouted through a different path with a higher Path MTU). For this reason, information learned from PMTUD expires after a certain amount of time to enable re-discovery of a potentially higher Path MTU.

> **Note**
>
> Another method for performing PMTUD, this time at the packetization layer, is described by RFC 4821, Packetization Layer Path MTU Discovery. This method is based on probes generated by the upper layer protocol (TCP for example), with increasing packet sizes in order to detect the Path MTU in a more reliable way. That method is still not widely deployed and is not covered in this book.

TCP MSS Clamping

While PMTUD provides a way to automatically adapt the size of TCP datagrams based on the discovered MTU, it still has disadvantages: it requires the loss and retransmission of one or more packets (depending on how many drops in MTU there are on the path), requires proper behavior from all filtering devices on the return path to deliver the ICMP messages back to the source, and may be broken if a device on the path to the destination clears the DF bit in IPv4 traffic. These last two points are particularly problematic, because they potentially involve devices that are outside the administrative domain of the sender.

One way to work around situations where PMTUD cannot be relied upon is to adjust the TCP Maximum Segment Size (MSS) on the fly, a method known as MSS clamping.

MSS Refresher

The MSS is a value advertised during the TCP 3-way handshake within the initial SYN and SYN ACK packets. It is advertised, as opposed to negotiated: each connection endpoint informs its peer of the largest TCP segment size that it is prepared to receive. The MSS represents the amount of data in the TCP packet, excluding the TCP header, options,and all lower-layer headers (IP and below), as illustrated in Figure 15-10. The typical length for the IP and TCP headers is 20 bytes each (though this value can be greater if IP or TCP options are present).

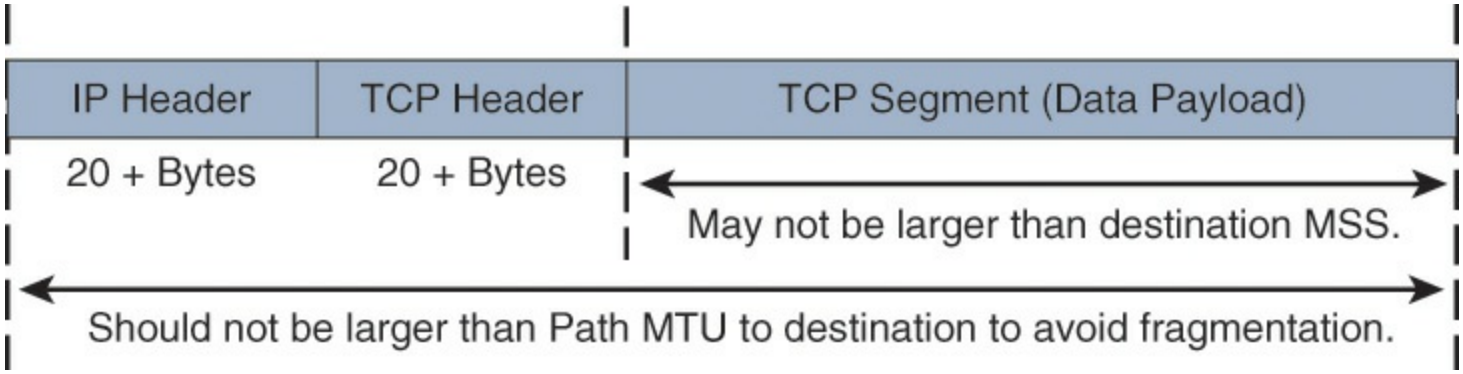


Figure 15-10 *TCP Maximum Segment Size (MSS)*

The advertised MSS is generally chosen based on the available buffer size for single TCP segments and the MTU on the output interface for the connection. The Path MTU to the remote endpoint (if known) should not be taken into account, as the MSS is relevant to inbound traffic, while the Path MTU is relevant to outbound traffic, which may be going through a different path (asymmetric routing situation).

If PMTUD signals to TCP that the Path MTU to a destination has decreased, the MSS that was recorded for that destination may need to be adjusted so that TCP packets remain within the limits of the Path MTU after encapsulation in IP.

Figure 15-11 illustrates the initial exchange of MSS during the 3-way handshake, as well as a simple example of automatic adjustment of the effective MSS triggered by PMTUD.

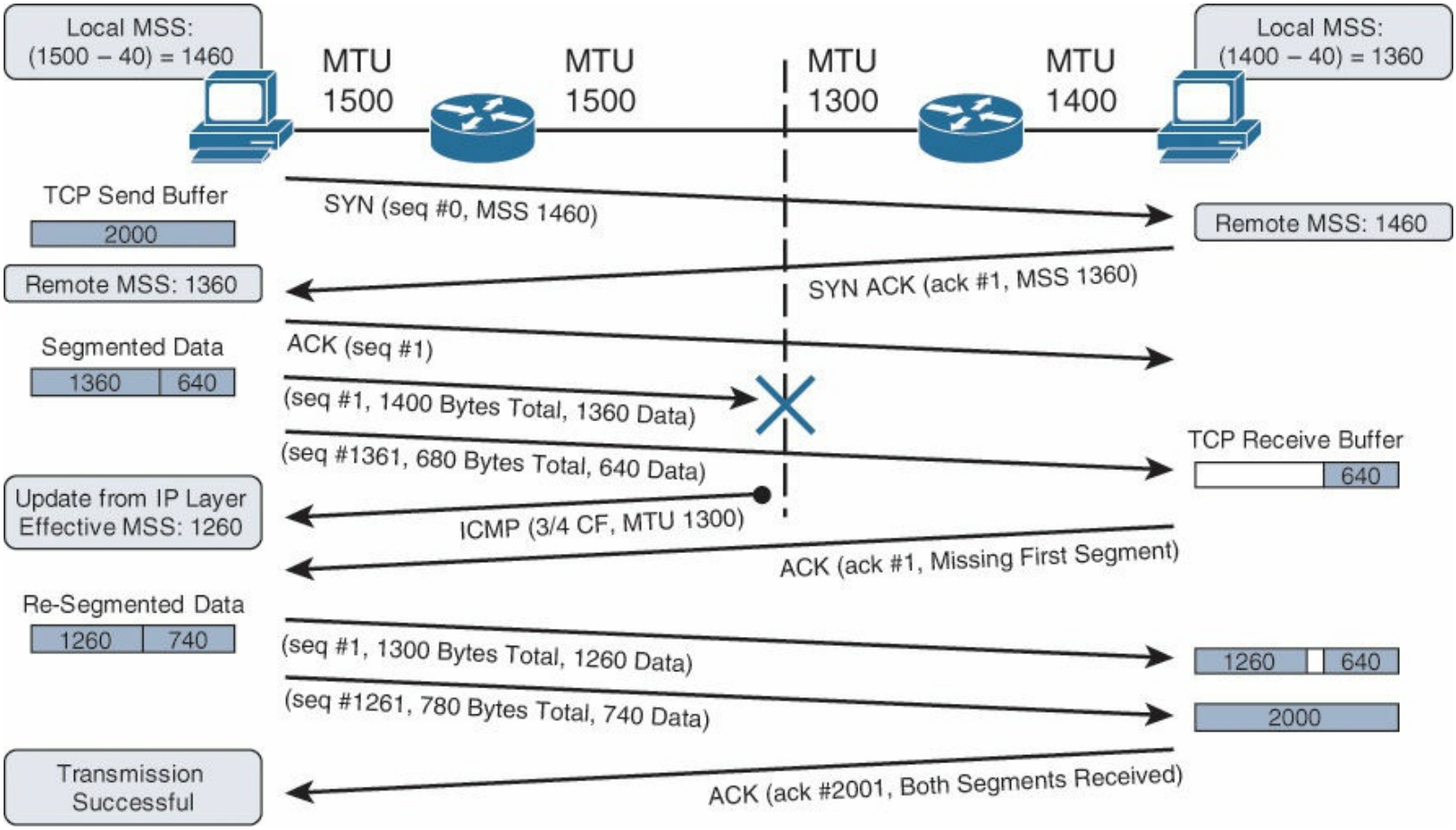


Figure 15-11 *TCP MSS exchange*

MSS Adjustment

In scenarios where PMTUD is not possible or reliable (for example, ICMP filtering by a firewall or IPS) or the Path MTU to some or all destinations is known to be lower than the output interface MTU on source hosts, it may be desirable to adjust the TCP MSS in both directions to a value low enough to ensure that PMTUD will not be required (assuming that the Path MTU remains above a certain value).

The TCP MSS in the SYN packets is controlled by the TCP endpoints, however Cisco routers can perform adjustment (also referred to as clamping) of the TCP MSS on the fly. This is accomplished by configuring the **ip tcp adjust-mss** *value* command on an interface traversed by the SYN packets (see the following example). Note that this command only takes effect for TCP connections going through the router, not those initiated by the router. For TCP connections originating or terminating on a router, the TCP MSS is controlled using the **ip tcp mss** *value* command,

Click here to view code image

```
interface FastEthernet0/0
 ip address 209.165.201.1 255.255.255.252
 ip tcp adjust-mss 1416
 ipv6 address 2001:DB8:FFF1::10/127
 ipv6 enable
```

The adjustment should be performed in a conservative way (lower is safer): if the command is configured on both the input and output interface traversed by the SYN packet, the MSS is adjusted to the lowest of the two configured values, and if the MSS in the SYN packet is already lower than the configured value, it is not modified.

> **Note**
>
> Initially the **ip tcp adjust-mss** command only affected TCP sessions going over IPv4. The behavior then changed in IOS 15.2(4)M and IOS-XE 15.3(1)S, in which the command was updated to also affect TCP sessions over IPv6. However, using a single MSS value for the two protocols did not prove to be practical due to the 20 extra bytes in the IPv6 header compared to IPv4 (the MSS had to be configured either 20 bytes too high for IPv6, or 20 bytes too low for IPv4). Because of this, IOS 15.3(3)M and IOS-XE 15.3(3)S reverted to the IPv4-only behavior of the original command, and a new **ipv6 tcp adjust-mss** command was introduced, enabling separate MSS adjustment for TCP sessions over IPv4 and IPv6.

Figure 15-12 shows a simple example of MSS clamping when the Path MTU is known to be lower in the middle of the path to the IPv4 destination because of the presence of a GRE/IPv6 tunnel (44 bytes of overhead). Since the Path MTU for IPv4 between the TCP endpoints is known to be 1456, the TCP MSS is clamped to a value of (1456 – 40) = 1416. Even though it is sufficient to configure the **ip tcp adjust-mss** command on only one of the routers (as illustrated) to adjust the MSS in both directions, it would be good practice to configure the command on both sides.

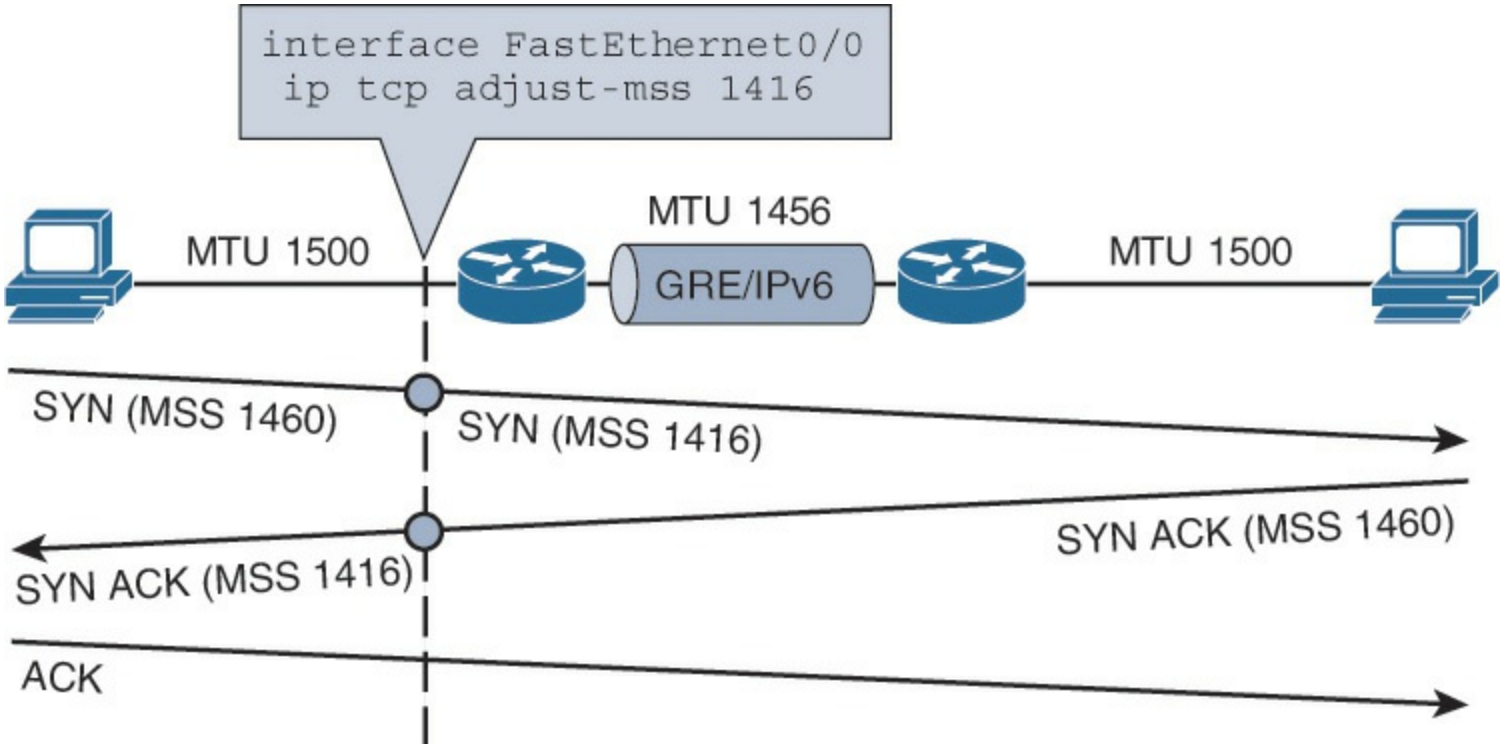


Figure 15-12 *MSS clamping*

IPsec Fragmentation and PMTUD

RFC4301, "Security Architecture for IP," mandates that IPsec implementations behave a specific way with regard to fragmentation and MTU.

With tunnel mode IPsec over IPv4, the DF bit in the outer IP header may be cleared, set, or copied from the inner IPv4 header. All three options must be available through configuration.

Every entry in the SA database contains a Path MTU value that represents the largest plaintext packet size (prior to encapsulation within IPsec) that it can protect. The initial value is derived from the output interface MTU minus the IPsec overhead, and the Path MTU is updated based on ICMP CF (ICMP 3/4 or ICMPv6 2/0) issued by downstream routers for IPsec packets. Upon receipt, the SPI in the original IPsec header is extracted from the portion of the dropped packet included in the ICMP message, and the Path MTU in the SA database entry for that SPI is updated after subtracting the IPsec overhead from the MTU in the ICMP packet.

When an outbound plaintext packet exceeds the Path MTU for the SA that will protect it, the encapsulating router handles it as follows:

- IPv4 traffic with the DF bit clear may be fragmented before or after encapsulation in IPsec tunnel mode. Both options must be available through configuration. IPsec transport mode will fragment only after encapsulation.
- IPv4 traffic with the DF bit set and IPv6 traffic are dropped, and an ICMP or ICMPv6 CF message is generated towards the source to enable PMTUD.

On Cisco routers, fragmentation before encapsulation (known as pre-fragmentation) or after (known as post-fragmentation) is controlled by the command **crypto IPsec fragmentation {before-encryption | after-encryption}.** Despite the name, the command applies to both Encapsulation Security Payload and Authentication Header. By default, IPsec will fragment before. The DF bit in the outer IP header is controlled by the **crypto IPsec df-bit {clear | set | copy}** command. The default is to copy the DF bit from the inner header. These commands may be configured globally and/or per

interface (with per-interface taking precedence).

> **Note**
>
> It is important to note that these commands only affect the behavior of the IPsec layer, not that of the tunnel layer (VTI or GRE/IPsec). Tunnel interfaces are hit before crypto in the forwarding path of the egress packet, and the tunnel MTU is a determining factor in deciding whether to pre-fragment or reject a packet. Please continue to "Fragmentation on Tunnels" to get a complete picture of fragmentation in tunnel-based configurations.

Figure 15-13 illustrates Encapsulation Security Payload tunnel mode fragmentation before and after encryption. The Interface MTU (1500) is the MTU of the physical interface that the IPsec traffic is sourced from. The Path MTU (1427) is initialized by subtracting the maximum IPsec overhead (73 bytes in this example) from the Interface MTU. Note the figure payload sizes are not to scale.

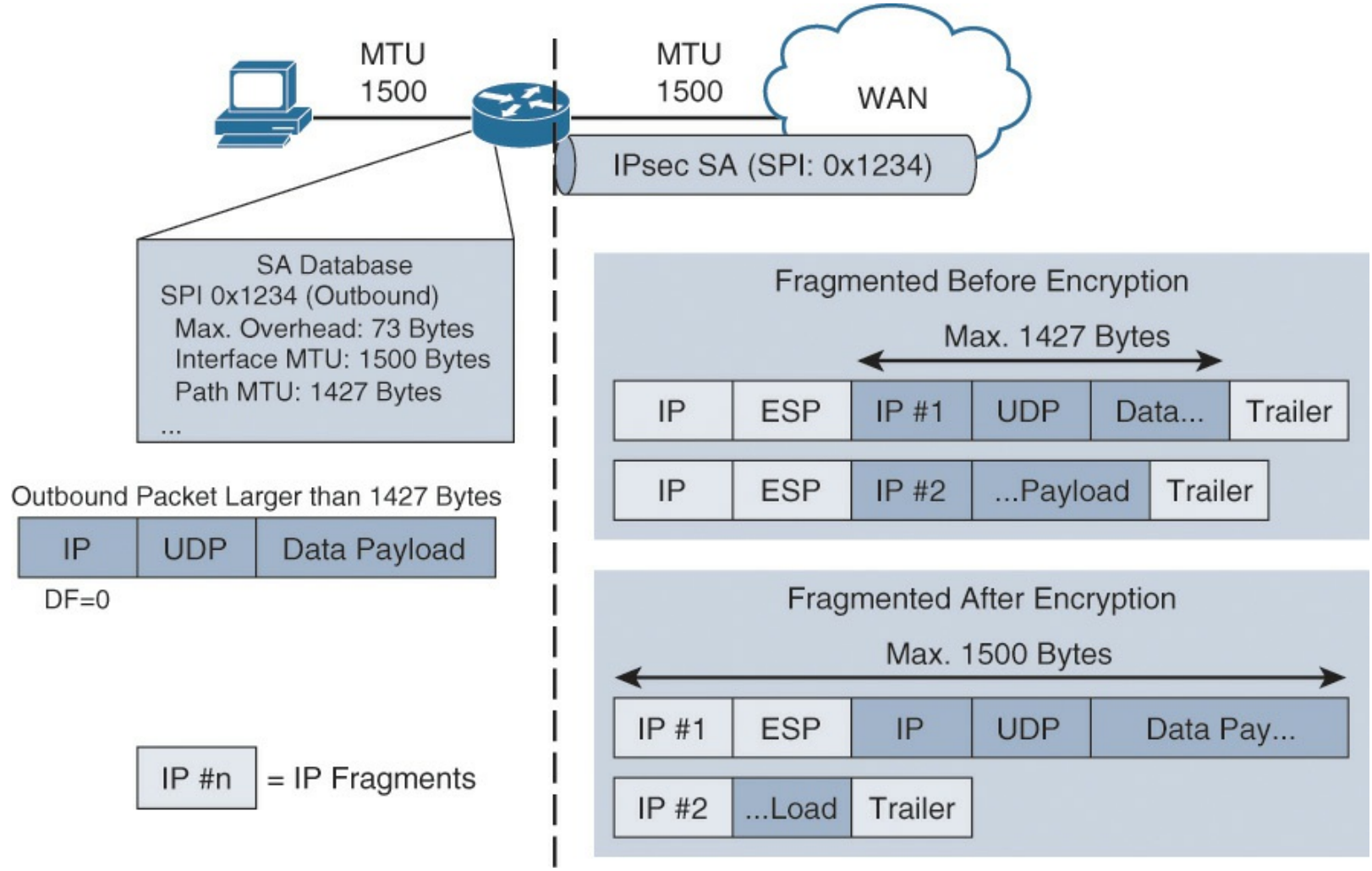


Figure 15-13 *Pre- and Post-Encryption Fragmentation*

The impact of fragmenting before or after IPsec will be discussed in "The Impact of Fragmentation" later in this section.

Figure 15-14 illustrates PMTUD on the source host in conjunction with IPsec PMTUD. If tunnel mode is used, this configuration requires setting or copying the DF bit to the outer IP header. The example shows IPv4 with the DF bit set, but the same logic applies to IPv6 with the implicit DF.

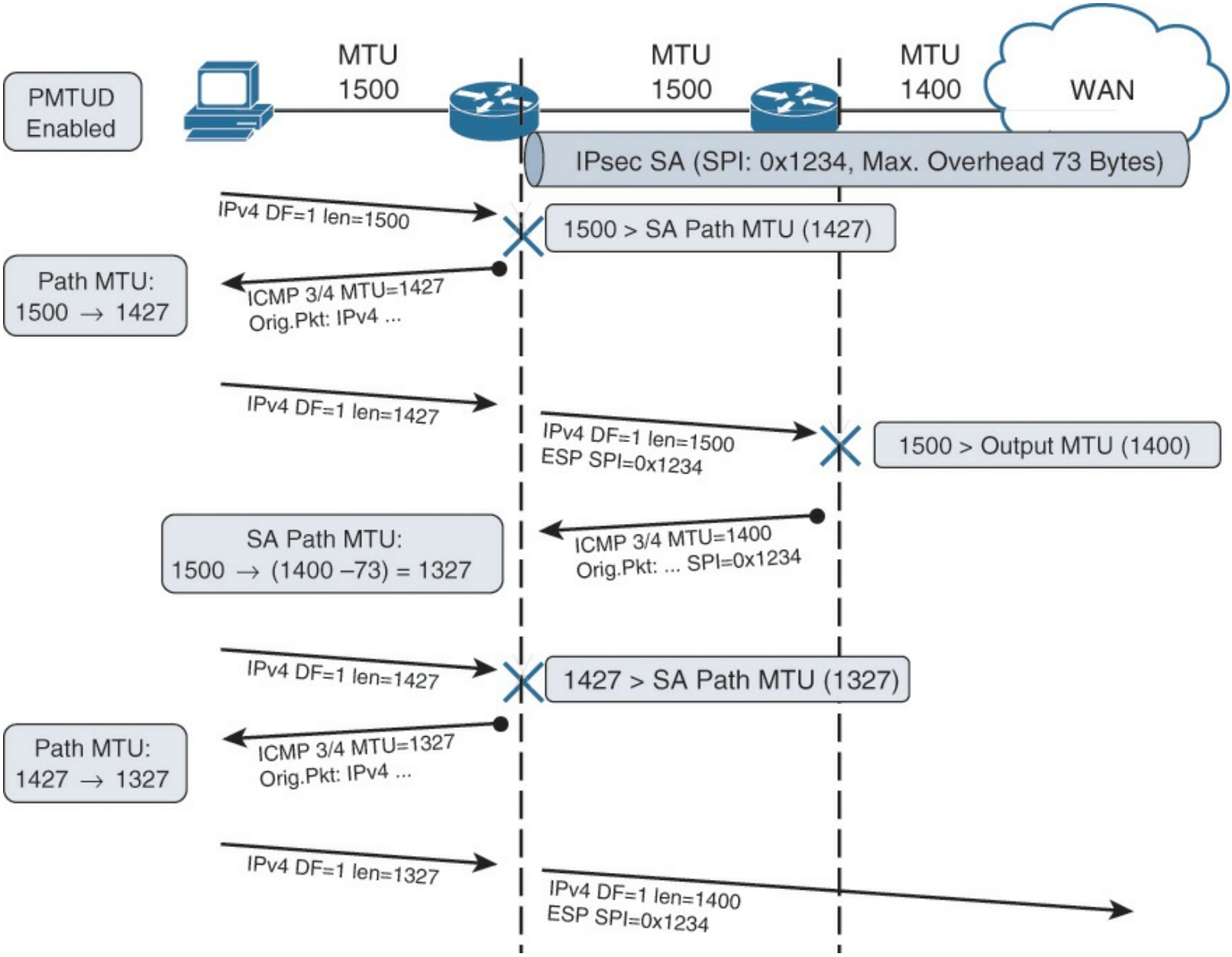


Figure 15-14 *PMTUD and IPsec*

One packet is lost when performing PMTUD at the entrance of the IPsec tunnel, plus two more packets for every drop in MTU on the path of the IPsec traffic: one to adjust the Path MTU on the SA, and another to adjust the Path MTU on the host. Further drops in MTU after the exit of the IPsec tunnel would only cause the loss of a single packet, but the ICMP CF message may or may not reach the source host, as it probably needs to be routed back through the IPsec tunnel. Since the source address of the ICMP packet is the router that dropped the traffic instead of the original destination, there is no guarantee that this address is also covered by an IPsec Security Association.

Until IOS 15.3(2)T and IOS-XE 15.3(2)S it was difficult to find out the SA Path MTU that was applied to plaintext traffic on IOS, as the output of **show crypto IPsec sa** only displayed the MTU and Path MTU as seen by IPsec on the transport network (including the encapsulation overhead). The following examples show an SA database entry that uses a physical interface with an MTU of 1500, but got an ICMP CF for IPsec traffic that indicated a path MTU of 1400.

The following example shows the output of **show crypto IPsec sa** collected on IOS 15.2(4)M. The SA Path MTU as seen by the plaintext traffic is not displayed and must be computed manually (or if the IPsec Security Association is bound to a VTI, it can be found in the output of **show ip interfaces** —see "Fragmentation on Tunnels" for details).

Click here to view code image

```
Router#show crypto ipsec sa | include mtu
     path mtu 1400, ip mtu 1500, ip mtu idb Ethernet0/0
```

The following example shows the same output collected on 15.3(3)M, now also including the plaintext MTU.

Click here to view code image

```
Router#show crypto ipsec sa | include mtu
     plaintext mtu 1342, path mtu 1400, ip mtu 1500, ip mtu idb Ethernet0/0
```

If the IPsec tunnel endpoints are IPv6, the path MTU to the peer address also appears in the IPv6 MTU table as illustrated in the following example.

Click here to view code image

```
Router#show crypto ipsec sa | include mtu
     path mtu 1400, ipv6 mtu 1500, ipv6 mtu idb Ethernet0/0

Router#show ipv6 mtu
 MTU      Since     Source Address      Destination Address
1400     00:00:11  2001:DB8:FFF1::10   2001:DB8:FFF1::2510
```

Fragmentation on Tunnels

Tunnel interfaces introduce an additional MTU check that takes place when the packet is routed through the tunnel, presenting an opportunity to take care of fragmentation before any encapsulation takes place—it is therefore crucial to understand the interaction between the different layers in order to reach a robust and optimal configuration. The following subsections detail the characteristics of the main modes of encapsulation used in this book.

IPsec Only (VTI)

A native IPsec tunnel interface (sVTI/dVTI) represents direct encapsulation into IPsec, thus by default the IP MTU of the tunnel is the same as the MTU on the associated IPsec Security Association. This default value can be overridden by explicitly configuring **ip mtu** or **ipv6 mtu**. The MTU for protocols other than IP (as displayed in show interfaces) is not relevant on VTI, since native IPsec can only protect IP traffic.

In the following example the tunnel is sourced from a loopback interface; that is, the IKE and IPsec traffic terminate locally on the IP address of the loopback. However, it is really the MTU of the physical output interface that matters. The IPsec code determines which physical interface is facing the remote peer and uses it as the basis for MTU calculations (Ethernet0/0 in the example). The same interface also shows up in the output of **show cef interface** for the VTI.

Click here to view code image

```
Router#show running-config interface Tunnel0
interface Tunnel0
 ip address 10.0.0.1 255.255.255.252
 tunnel source Loopback1
 tunnel mode ipsec ipv4
 tunnel destination 209.165.200.225
 tunnel protection ipsec profile default
end
```

```
Router#show crypto ipsec sa | include mtu
     path mtu 1500, ip mtu 1500, ip mtu idb Ethernet0/0

Router#show interface Tunnel0 | include MTU
  MTU 17878 bytes, BW 100 Kbit/sec, DLY 50000 usec,
  Tunnel transport MTU 1438 bytes

Router#show ip interface Tunnel0 | include MTU
  MTU is 1438 bytes

Router#show cef interface Tunnel0 | include output interface
  Real output interface is Ethernet0/0
```

As with all tunnel interfaces, the packet length is checked against the MTU of the interface before the encapsulation takes place. For VTI, it is before the packet is handed over to the crypto engine; this means that VTI always performs pre-fragmentation regardless of the **crypto IPsec fragmentation** setting, which only applies once the packet has hit the crypto layer. The only case in which a VTI may do post-fragmentation is if the IP MTU of the VTI is hardcoded to a value that is higher than the IPsec Security Association PMTU (due to misconfiguration, or a drop of MTU on the path that was discovered through PMTUD).

Figure 15-15 shows a block representation of VTI encapsulation and the MTU values encountered during processing of the packet.

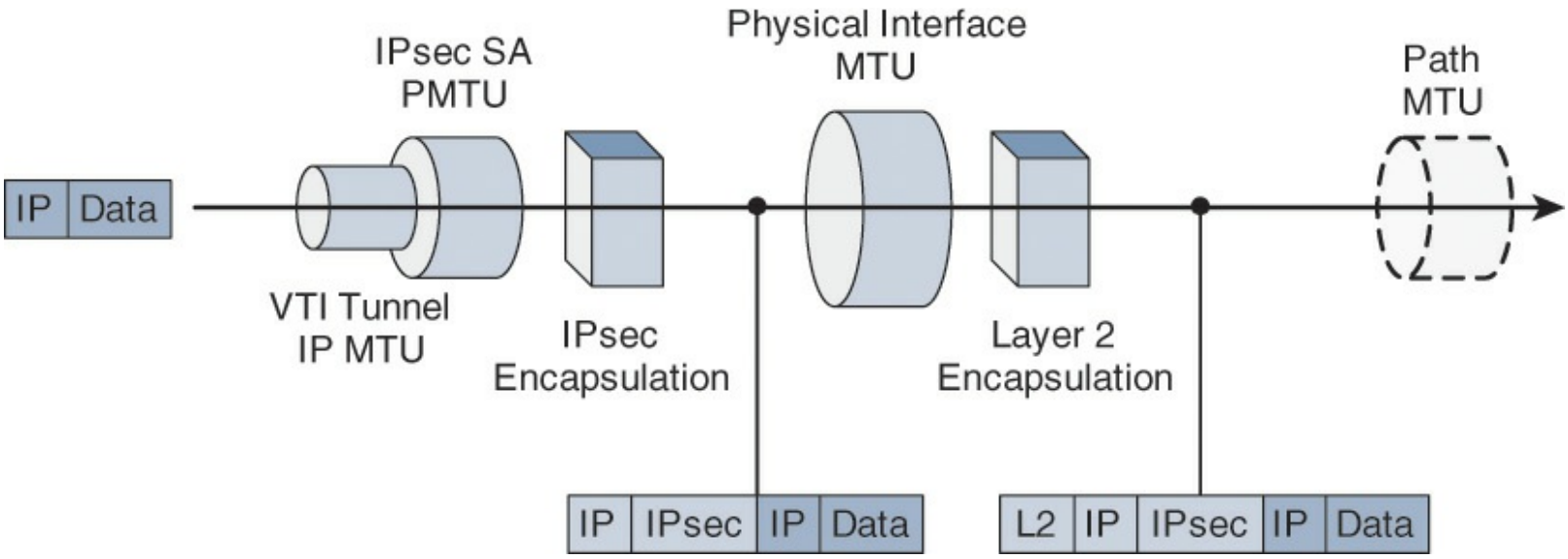


Figure 15-15 *VTI encapsulation with MTU*

GRE Only

The default MTU of a GRE tunnel interface without IPsec is equal to the IP MTU of the output interface minus the IP and GRE encapsulation overhead. Just as with VTI, the tunnel code is aware of the actual output interface and will select the correct physical IP MTU as a basis, even if sourced from a loopback. Since GRE is capable of transporting any kind of protocol including non-IP, the tunnel code will adjust the MTU for all protocols (as reported in the **show interfaces** command).

The smallest possible GRE overhead is 24 bytes (GRE/IPv4 without optional fields) and the largest is 56 bytes (GRE/IPv6 with tunnel key, checksum and sequence numbering), without counting optional extension headers. The following example shows a GRE/IPv6 tunnel with a tunnel key configured, amounting to 48 bytes of overhead in total.

Click here to view code image

```
Router#show running-config interface Tunnel0
interface Tunnel0
 ip address 10.0.0.1 255.255.255.252
 tunnel source Ethernet0/0
 tunnel mode gre ipv6
 tunnel destination 2001:DB8:FFF1::2510
 tunnel key 1
end

Router#show interface Tunnel0 | include MTU
  MTU 1452 bytes, BW 100 Kbit/sec, DLY 50000 usec,
  Tunnel transport MTU 1452 bytes

Router#sh ip interface Tunnel0 | include MTU
  MTU is 1452 bytes
```

The IP MTU on the tunnel can be adjusted with the **ip mtu** or **ipv6 mtu** commands. Both can be configured simultaneously, because GRE allows for dual-stack (IPv4 and IPv6 inside the same GRE tunnel). The MTU for other protocols cannot be adjusted (the **mtu** command is not allowed on tunnel interfaces).

Figure 15-16 shows a block representation of GRE encapsulation and the MTU values encountered during processing of the packet.

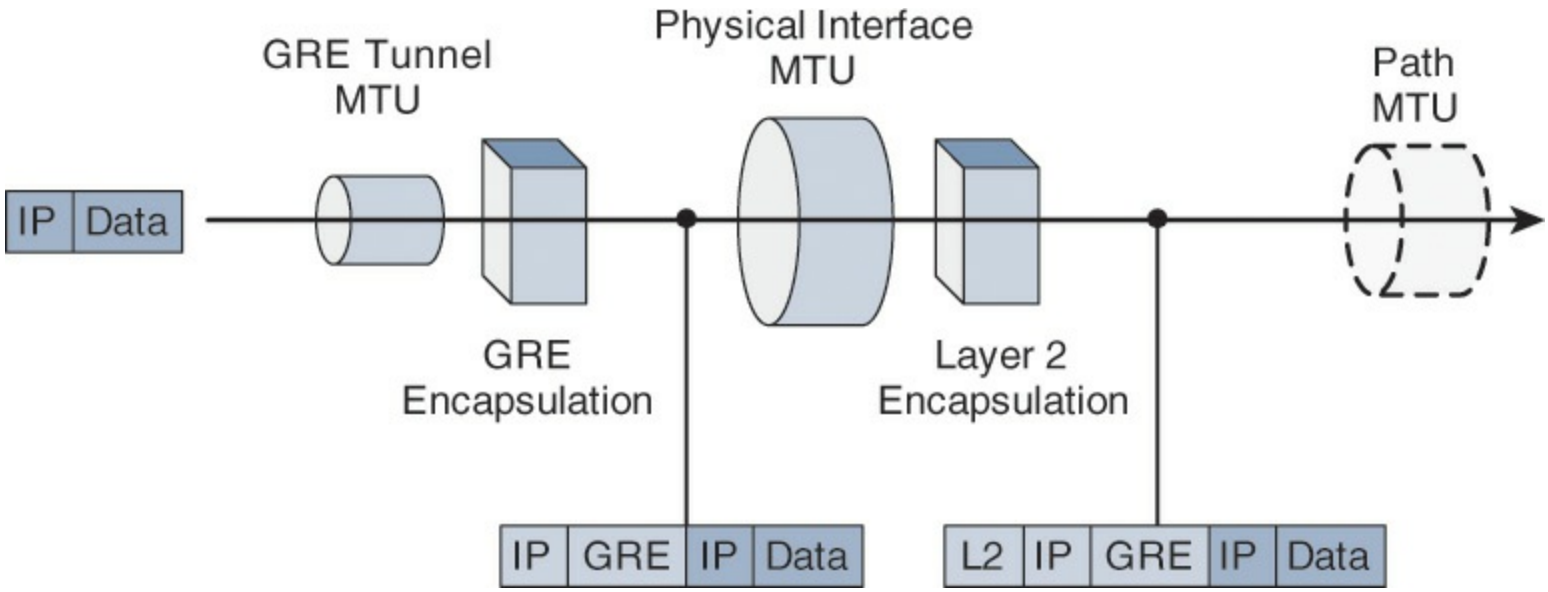


Figure 15-16 *GRE encapsulation with MTU*

An important characteristic of GRE over IPv4 is that by default it clears the DF bit in the outer IP header, so the check against the physical interface MTU may result into fragmentation even if the inner IP packet does have the DF bit set. The solution to this problem is covered in "Tunnel PMTUD."

GRE over IPsec

Adding IPsec on top of a GRE interface (with the **tunnel protection** command) adds a layer of IPsec encapsulation right after the creation of the GRE packet. Just like the behavior of VTI, the MTU on the GRE tunnel is bound to that of the IPsec Security Association, so that a drop in the SA Path MTU is reflected on the MTU of the GRE tunnel. The GRE interface also allows for manually setting the IP MTU on the GRE interface to ensure that all produced GRE packets will be small enough to fit inside the IPsec plaintext MTU.

Figure 15-17 shows a block representation of GRE/IPsec encapsulation and the MTU values encountered during processing of the packet.

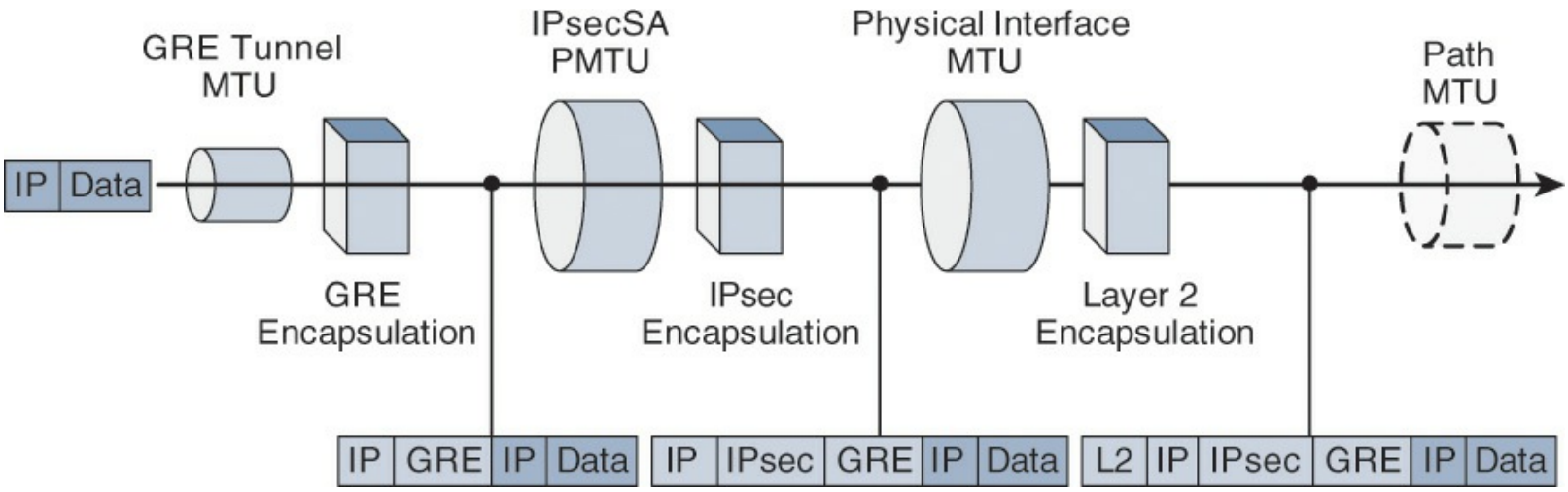


Figure 15-17 *GRE over IPsec MTU*

Two options exist in order to properly set the IP MTU on the GRE tunnel. The first is to compute the maximum GRE/IPsec overhead, subtract it from the physical interface MTU, and configure it using **ip mtu** or **ipv6 mtu** on the tunnel interface. This is usually done in conjunction with adjusting the TCP MSS (see "TCP MSS Clamping" earlier in this section). The second option is Tunnel PMTUD (see below).

Tunnel PMTUD

PMTUD discovery allows for tunnel interfaces to automatically adjust the MTU based on received ICMP CF messages. To enable a tunnel interface to automatically adjust MTU based on PMTUD, ICMP unreachable messages must be received on the source interface of the tunnel. This requires any access control policy, including an ACL applied on ingress to the tunnel source, to permit ICMP 3/4 or ICMPv6 2/0 messages (also known as ICMP CF messages).

A GRE IPsec tunnel interface will by default not copy the DF bit from the protected IP header to the outer GRE header. This in some instances can cause issues due to fragmentation, but it also results in PMTUD not functioning, as traffic must have the DF bit set for this to occur. To enable PMTUD on a GRE IPsec tunnel interface, the **tunnel path-mtu-discovery** command must be enabled (the default behavior is not enabled). This will enable copying of the DF bit from the inner protected IP header to the outer GRE IP header.

Once PMTUD has been enabled on a tunnel interface, it can be verified as illustrated in the following example.

Click here to view code image

```
Router#show interface tunnel 1 | include Path MTU
  Path MTU Discovery, ager 10 mins, min MTU 92
```

There are a number of options for the PMTUD command. The following example illustrates the age timer and the minimum MTU size.

Click here to view code image

```
Router(config)#interface tunnel 1
Router(config-if)#tunnel path-mtu-discovery ?
  age-timer  Set PMTUD aging timer
```

```
    min-mtu    Min pmtud mtu allowed
    <cr>

  Router(config-if)#tunnel path-mtu-discovery age-timer ?
    <10-30>   Aging time
    infinite  Disable pathmtu aging timer

  Router(config-if)#tunnel path-mtu-discovery min-mtu ?
    <92-65535>  Bytes
```

VTI will by default adjust the tunnel MTU when ICMP CF messages are received. The command **tunnel path-mtu-discovery** has no effect on a VTI. A VTI will by default copy the DF value in the IP header from the protected IP packet to the outer IP header. In other words, out-of-the-box VTI will always perform PMTUD as long as the inner protected traffic is marking the DF bit in the IP header.

As the **tunnel path-mtu-discovery** command has no effect for a VTI, if the behavior needs to be changed, then clearing the IP DF bit is possible on the interface. This is achieved by using the **crypto IPsec df-bit clear** command on the VTI or globally.

For either VTI or GRE IPsec, if PMTUD is enabled with the **tunnel path-mtu-discovery** and the tunnel interface has the **ip mtu** command applied, then the lowest MTU will take precedence. So if the dynamically obtained MTU via PMTUD is lower than the configured IP MTU, the PMTUD MTU will be used; otherwise the IP MTU will be used.

The Impact of Fragmentation

There are several issues that make IP fragmentation undesirable. When traffic requires fragmentation, there is an increase in CPU and memory overhead to fragment an IP datagram. This holds true for the sender as well as for a router in the path between a sender and a receiver. As the creation of fragments simply involves generating a fragment header and copying the original datagram into the fragment, it can be achieved fairly efficiently because all the information needed to create the fragments is immediately available.

Fragmentation causes more overhead for the receiver when it reassembles the fragments, because the receiver must allocate memory for the arriving fragments and reconstruct all of them back into one datagram once all of the fragments are received. Reassembly on a host is generally not considered a problem, because the host in many circumstances has the time and memory resources to devote to this task.

Reassembly is very inefficient on a router whose primary job is to forward packets as quickly as possible. A router is not designed to hold on to packets for any length of time. Also a router that does reassembly chooses the largest buffer available (18KB) with which to work because it has no way to know the size of the original IP packet until the last fragment is received. Reassembly of packets fragmented after encryption on a router can cause the packets to arrive out of order at the destination host; also the reassembled packet can fall out of the anti-replay window.

In many instances when fragmentation is occurring, network performance will be severely degraded, with reports of slow connectivity and intermittent access.

Another fragmentation issue involves how dropped fragments are handled. If one fragment of an IP datagram is dropped, then the entire original IP datagram must be resent, and it will also be fragmented.

Many network devices, such as Network Address Translation (NAT) gateways or load balancers, might have trouble processing IP fragments correctly. If the IP fragments are out of order, a NAT gateway blocks the non-initial fragments because they do not carry the information that would match the packet filter. This would mean that the original IP datagram could not be reassembled by the receiving host. In many cases, simple access control lists (ACL) do not reconstruct fragments, so they do not apply policy on fragment traffic, either allowing or denying all fragments.

Summary

IPsec always incurs a form of overhead; however, depending on the algorithms used and the method of encapsulation, the amount of overhead can vary. Encapsulation Security Payload, Authentication Header, GRE and the modes of IPsec all have defined overheads.

Depending on the overhead generated, the MTU for tunnel interfaces is automatically adjusted. This can be manually configured or dynamically adjusted using PMTUD. There are considerations when using PMTUD for both GRE IPsec and VTI.

One of the side effects of IPsec overhead is fragmentation; this is the nemesis of many network architects and engineers who are tasked with understanding why network connectivity is degraded. Enabling PMTUD or manually setting the IP MTU on a tunnel interface can mitigate issues resulting from fragmentation.

References

https://tools.ietf.org/html/rfc791 Internet Protocol

https://tools.ietf.org/html/rfc1191 Path MTU Discovery

https://tools.ietf.org/html/rfc1981 Path MTU Discovery for IP version 6

https://tools.ietf.org/html/rfc2401 Security Architecture for the Internet Protocol

https://tools.ietf.org/html/rfc2403 The Use of HMAC-MD5-96 within Encapsulation Security Payload and Authentication Header

https://tools.ietf.org/html/rfc2404 The Use of HMAC-SHA-1-96 within Encapsulation Security Payload and Authentication Header

https://tools.ietf.org/html/rfc2410 The NULL Encryption Algorithm and Its Use With IPsec

https://tools.ietf.org/html/rfc2451 The Encapsulation Security Payload CBC-Mode Cipher Algorithms

https://tools.ietf.org/html/3602 The AES-CBC Cipher Algorithm and Its Use with IPsec

https://tools.ietf.org/html/rfc3566 The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec

https://tools.ietf.org/html/3686 Using Advanced Encryption Standard (AES) Counter Mode with IPsec Encapsulating Security Payload

https://tools.ietf.org/html/rfc4106 The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload

https://tools.ietf.org/html/rfc4301 Security Architecture for the Internet Protocol

https://tools.ietf.org/html/rfc4302 IP Authentication Header

https://tools.ietf.org/html/rfc4303 IP Encapsulating Security Payload

https://tools.ietf.org/html/rfc4309 Using Advanced Encryption Standard (AES) CCM Mode with

IPsec Encapsulating Security Payload

https://tools.ietf.org/html/rfc4494 The AES-CMAC-96 Algorithm and Its Use with IPsec

https://tools.ietf.org/html/rfc4543 The Use of Galois Message Authentication Code (GMAC) in IPsec Encapsulation Security Payload and Authentication Header

https://tools.ietf.org/html/rfc4821 Packetization Layer Path MTU Discovery

https://tools.ietf.org/html/rfc4868 Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with Ipsec

Part VIII: Migration to IKEv2

Chapter 16. Migration Strategies

Introduction to Migrating to IKEv2 and FlexVPN

Over the last three to four years, many customers have realized the benefits of using IKEv2 and started to migrate to VPN architectures that make use of the benefits of IKEv2 and FlexVPN.

In many instances, customers will make the transition at the same time they upgrade hardware. The rationale is that older hardware does not support the IKEv2 protocol, so the interruption caused by upgrading hardware and migrating to IKEv2 can be combined, minimizing the impact.

VPN architectures already in service are a critical component of any ICT system, allowing remote workers to connect and/or interconnecting remote offices or data centers. Thought and care are required in the migration of VPN architectures. Downtime should be minimized to avoid disruption to services. Due-diligence is needed to ensure that the new service provides the same and/or additional functionality as the previous solution. Functionality could be as simple as providing IP connectivity to clients or as complex as requiring dual stack (IPv4 and IPv6) with multiple VPN gateways with clients tracking connectivity to each gateway.

No two VPN architectures will be the same, so there is no universal solution when migrating from using IKEv1 to IKEv2. For this reason the concepts described within this chapter are based on high-level concepts and should be taken as a guide.

Consideration when Migrating to IKEv2

If there are plans to migrate a current VPN solution to IKEv2 or to have a new deployment use IKEv2, the following aspects need to be addressed.

Hardware Limitations

Cisco ISR Generation 2 (G2) routers (800, 1900, 2900, 3900), ISR 4000 series (43XX and 44XX), CSR1000V, and ASR1000 hardware include the support for IKEv2 and FlexVPN.

ISR Generation 1 routers do not support IKEv2.

Note that all ISR G2, ISR 4000 series, and CSR1000V platforms support next-generation encryption (NGE) algorithms for both control plane (IKEv2) and data plane (IPsec). Only ASR octeon-based platforms (ASR1001-X, ASR1002-X, Embedded Services Processor-100, and Embedded Services Processor-200) support NGE for both control plane and data plane. Older ASR 1000 platforms (non-octeon) support NGE only in the control plane.

In many VPN deployments not all the hardware will have the ability to support IKEv2. In this case the hardware will need to be upgraded. This is a perfect time to have the hardware upgrade coincide with transitioning the configuration from IKEv1 to IKEv2, rather than installing IKEv2-compatible hardware and configuring only IKEv1.

Current VPN Technology

Different VPN technologies allow for different features to be used. FlexVPN can be used to replace any overlay VPN technology, including site-site, remote-access, hub-and-spoke, and spoke-spoke. Specific technologies such as EzVPN or DMVPN can be migrated to FlexVPN. Specific considerations depend on which specific features are used.

Most of the features that are available within EzVPN are also available in FlexVPN, except for:

- **Auto-update**, an automated upgrading mechanism for software and firmware.
- **URL configuration**, a mode-configuration attribute pushed from a URL from the VPN server to the EzVPN remote device. The URL contains the configuration information that the remote device has to download and apply to the running configuration.

If these features are required, please contact your Cisco account team for guidance.

Other features that are available with EzVPN, such as allocation of DHCP, WINS, and domain-name, are all available and configurable using FlexVPN server. If a username and password are used as the method of authentication, this can be achieved with EAP credentials locally on the FlexVPN client.

The following example illustrates configuring local EAP credentials.

Click here to view code image

```
Router(config)#crypto ikev2 profile default
Router(config-ikev2-profile)#authentication local eap ?
  gtc       eap method gtc credentials
  md5       eap method md5 credentials
  mschapv2  eap method mschapv2 credentials
  <cr>
Router(config-ikev2-profile)#authentication local eap md5 ?
  password  EAP password
  username  EAP username
```

Routing Protocol Selection

If the current VPN architecture uses a routing protocol and the new VPN architecture is required to integrate into this existing routing architecture, care should be taken to ensure that disruption does not occur. In most instances separate autonomous systems are recommended for each group of VPN devices. So an IKEv1 deployment would use one autonomous system, and an IKEv2 deployment would use another. This allows a clean separation between separate instances. This isn't a strict rule, but has resulted in smooth deployments.

IKEv2 allows routing to occur over the IKE exchange, which allows an IKEv1 deployment to run a routing protocol and the new IKEv2 deployment to use IKEv2 routing. The benefits of using IKEv2 routing is that it is not a chatty protocol; it can easily be added to existing implementations and redistributed into the current autonomous system, because IKEv2 routes are installed as static routes. IKEv2 routing also provides the ability to tag prefixes learned via IKEv2. This can be useful in cases where IKEv2 routing is redistributed into routing protocols that will then make decisions based on the tagged prefixes. Tagging can be very useful for deployments where routing loops can occur. Tagging prefixes and using route maps to limit the redistribution of certain prefixes can prevent routing loops and suboptimal routing.

The administrative distance on prefixes learnt over IKEv2 can be set locally, this allows for tunnel interfaces which use IKEv2 routing to have the local administrative distance set to a higher (less

preferable) metric than the routing protocol currently being used over a tunnel interface using IKEv1. This allows for the Routing Information Base (RIB) to be populated with the prefixes; however, the active prefixes will be learnt using the routing protocol running over the IKEv1 tunnel and not the IKEv2 tunnel, until the IKEv1 tunnel can be shut down and the corresponding IKEv1 prefixes removed.

Restrictions When Running IKEv1 and IKEv2 Simultaneously

In Cisco IOS, an IKEv1 and an IKEv2 SA cannot exist within the same IPsec profile on a tunnel interface. A tunnel interface has to run either IKEv1 or IKEv2; it cannot run both simultaneously. However, separate tunnel interfaces can be configured from and to the same peers, with one being protected by IKEv1 and the other by IKEv2.

A crypto map can have entries that use both IKEv1 and IKEv2 profiles; however, these must be referenced in separate entries. If a crypto map entry is configured for both IKEv1 and IKEv2 profiles, the error in the following example will be seen.

Click here to view code image

```
Router(config-crypto-map)#set isakmp-profile IKEv1
Router(config-crypto-map)#set ikev2-profile IKEv2
% WARNING: Configuring Ikev2 profile will remove the Ikev1 profile configured
under this ipsec profile
```

Because of the conditions described previously, it is always advised to run separate tunnel interfaces, with each running either IKEv1 or IKEv2. This allows for simple configurations in which each tunnel interface clearly describes the IKE version, resulting in clarity when monitoring and troubleshooting.

Current Capacity

There is a fixed limit on the number of IPsec sessions that can coexist on any given hardware.

The following example illustrates the method to view the number of IPsec sessions that are available on a given platform. Care should be taken so that the capacity for IKE and IPsec sessions is not exceeded if multiple tunnels are needed for IPsec sessions to new devices.

Click here to view code image

```
Router#show crypto eli
Hardware Encryption : ACTIVE
 Number of crypto engines = 3

 CryptoEngine Onboard VPN details: state = Active
 Capability    : IPPCP, DES, 3DES, AES, GCM, GMAC, IPv6, GDOI, FAILCLOSE, HA

 CryptoEngine Software Crypto Engine details: state = Active
 Capability    : IPPCP, DES, 3DES, AES, SEAL, GCM, GMAC, RSA, IPv6, GDOI,
FAILCLOSE, HA

 IKE-Session   :     2 active,  2100 max, 0 failed
 IKEv2-Session :     2 active,  2100 max, 0 failed
 DH            :     1 active,  1050 max, 0 failed
 IPSec-Session :     0 active,  1000 max, 0 failed
```

When migrating from IKEv1 to IKEv2, one needs to monitor the capacity on a device if multiple tunnels are to coexist on it to ensure that the onboard limits are not exceeded during the migration.

This can be minimized by restricting small groups of peers to connect using both IKEv1 and IKEv2 simultaneously. Allowing large groups of devices to connect simultaneously has potential for exhausting all available crypto engine resources because the resources required per peer double. If, for example, the current load was 70% of the full capacity, it could be prudent to migrate only a limited number of hosts at any given time so that capacity does not exceed 80%.

Factors such as the number of tunnels that can be set up per second should be taken into account. If the new IKEv2 SAs will use a Diffie-Hellman group that differs from the current IKEv1 SAs, care should be taken to ensure that the additional cryptographic load can be managed. Please consult Cisco Technical Assistance Center (TAC) or Advanced Services (AS) should professional guidance be required.

IP Addresses

If using FlexVPN server and allocating clients' IP addresses, one needs to investigate whether there is enough capacity to allocate IP addresses for the new IP address pool. This is usually the largest allocation of IP addresses for a new deployment.

Furthermore, if a new piece of hardware running IKEv2 is going to be integrated into the current architecture, is current capacity sufficient to add IP addresses for the physical and tunnel interfaces?

Software

In some instances, devices running older versions of IOS or IOS-XE software might not actually support IKEv2 or other newer features that are required. In this instance, software can be upgraded on these devices, and then new IKEv2 protected traffic can be tested.

Another approach is to perform the software upgrade first. Once stability has been confirmed, IKEv2 can be implemented. This minimizes the chances that any degradation of performance caused by the upgraded software would be blamed on the IKEv2 component.

Amending the VPN Gateway

When making any changes on a live system, take care to ensure that disruptions to service do not occur. Adding additional tunnel interfaces correctly should not cause any disruption to traffic; however, care should be taken that the configuration is correct. For this reason, it is advisable to recreate the procedure in a lab first or consult with Cisco Advanced Services.

The initial contact feature, where any new session will tear down previous sessions with the same IKE identity, is only active with the same protocols. So if a peer had both IKEv1 and IKEv2 sessions with another system, these could coexist without initial contact coming into force.

Global IKE and IPsec Commands

Many commands relating to IPsec are global and can be used by IPsec Security Associations created by either IKEv1 or IKEv2. These settings are for global IPsec attributes and will be relevant for IPsec Security Associations created by both IKEv1 and IKEv2. The following example illustrates some of the global IPsec commands available.

Click here to view code image

```
Router(config)#crypto ipsec ?
  client                Configure a client
```

```
  df-bit                Handling of encapsulated DF bit.
  fragmentation         Handling of fragmentation of near-MTU sized packets
  ipv4-deny             Configure global ipv4 deny policy.
  nat-transparency      IPsec NAT transparency model
  optional              Enable optional encryption for IPSec
  profile               Configure an ipsec policy profile
  security-association  Security association parameters
  transform-set         Define transform and settings

Router(config)#crypto ipsec security-association ?
  dummy      Enable transmitting dummy packets
  ecn        Handling of ECN bit
  idle-time  Automatically delete IPSec SAs after a given idle period.
  lifetime   security association lifetime
  replay     Set replay checking
```

If SNMP monitoring is enabled for IKE or IPsec, the messages will be generated by either IKEv1 or IKEv2. There is no method to differentiate between either IKE protocol.

FlexVPN Features

FlexVPN brings a plethora of new features. When looking to move from an IKEv1 to IKEv2 deployment, understand the features that FlexVPN offers and evaluate them to ensure that the transition brings the most benefits.

The features that FlexVPN introduces could influence purchasing decisions if hardware is also upgraded. Features such as Tunnel Pivot, which allows a tunnel interface to amend the tunnel source based on a tracked event, or Backup Peers, where the tunnel destination can dynamically change, can potentially result in hardware configurations requiring fewer interfaces that previously needed.

FlexVPN combines a number of previous Cisco VPN technologies. Before FlexVPN was available, a number of hardware devices might have been required to terminate different types of VPN topologies, while now a single hardware device can be used.

Although not specific to FlexVPN, if the current VPN architecture uses crypto maps to protect traffic, the decision to move to using tunnel interfaces should be considered. A benefit of using tunnel interfaces when migrating from IKEv1 to IKEv2 is that only a single IPsec Security Association is created for the tunnel interface. With a crypto map, on the other hand, an IPsec Security Association is created for every entry within the crypto map's access control list. In this instance there is less chance of exceeding the maximum number of IPsec Security Associations.

Familiarization

Any individual who is experienced with IKEv1 will require some degree of retraining when migrating a VPN architecture from IKEv1 to IKEv2.

The architects and designers of the VPN architecture should be familiar with the features that IKEv2 brings. Such features as high availability and third-party support should be understood to ensure that technologies are aligned to business requirements.

Staff who will be operating and monitoring the new architecture should be familiar with the operations of IKEv2. Knowing how the protocol operates will assist greatly in any troubleshooting or diagnostic scenario. Because Cisco TAC worked closely with the development team within Cisco, there are many commands and debug options that were not available in IKEv1. Support staff should

be made aware of such tools to maximize productivity and minimize downtime.

Client Awareness

When a remote access architecture is in use, a move from IKEv1 to IKEv2 could require users to get a new version of code for the VPN client. In this instance, the new client software will need to be installed and checked for compatibility issues with the operating system; it will also be necessary to ensure that it operates as intended.

Public Key Infrastructure

For many customers that are moving to IKEv2 and have never previously had a requirement to use PKI, PKI is a new concept. There is a common assumption that setting up an internal PKI is a simple task, and this effort is not factored into the scope and budget for projects that involve migrations or new deployments using IKEv2. On the contrary, setting up a well-designed and architectured PKI can consume as much time (or even more) than as designing the VPN itself!

Cost can be a factor when designing a PKI. Hardware Security Modules (HSM) allow for governance around the storage and operation of key material. HSM are relatively expensive pieces of equipment and require specialist skill. This can come as a shock to customers who are required to use a HSM when it is specified in a standard or accreditation that they are governed by.

The following questions and considerations should be addressed when using IKEv2 with a PKI:

- Should the current authentication mechanism not include PKI, and if PKI is introduced, will there be skilled individuals to design, architect, and support this?
- When PKI is used as the method of authentication, will the existing capacity of the PKI be exceeded?
- Can the existing PKI issue certificates for use with IPsec VPNs?
- Is there a cost for obtaining certificates?
- Will the installation of additional certificates on a client machine cause another PKI-based service to use the incorrect certificate?
- Could the addition of revoked certificates cause disruption to other functions, consuming PKI services such as CRLs?
- Is there accurate time available to ensure that validations do not fail or that expired certificates can’t be used?

Since nearly all remote-access IKEv2 deployments use EAP and/or digital signatures for authentication, PKI is a critical component. For this reason, care must be taken to ensure that the PKI is available and redundant. Outages can occur if a CRL is not published in a timely manner, or if OCSP checks fail. Having accurate and assured time is a mandatory requirement when using PKI; this is another factor to consider when clients are on an untrusted network.

Internet Protocol Version 6

In many circumstances when a new VPN gateway is deployed, IPv4 and/or IPv6 connectivity is required. It is now becoming common for transport networks to support only IPv6, so even if the overlay network is running solely IPv4, it can be prudent to enable both IPv4 and IPv6 on the VPN gateway for the transport network to ensure clients using any transport network can successfully connect.

If there is a requirement for both IPv4 and IPv6 to be transported within the same IPsec Security Association, then dual stack can be used. Auto tunnel mode can be used to support clients running IPv4 or IPv6 transport protocols, with the ability to dynamically create an IPv4 or IPv6 virtual-access interface on demand.

Mixed mode allows for IPv4 networks to be transported over an IPv6 transport network when using VTI. GRE allows for IPv4 and IPv6 prefixes to be transported over an IPv4 or IPv6 transport network.

When using DNS with IPv4 and IPv6, ensure that the A record and AAAA record are available. This is a common oversight when configuring IPv6-enabled clients.

Authentication

IKEv1 and IKEv2 use similar, but not identical authentication methods. While IKEv1 provides pre-shared key, RSA signatures, RSA nonces, and username/password when using extended authentication (X-Auth), IKEv2 supports authentication using pre-shared keys, RSA signatures, ECDSA signatures, and EAP.

If the current authentication method used is not available with IKEv2, then a strategy is required to provide a new means of authentication. For example, this may occur when implementing a PKI for the new IKEv2 deployment or migrating users from extended authentication to using EAP (which also requires a PKI).

High Availability

If a load balancer is currently used to distribute load between IKEv1 gateways and all clients connect to a single (virtual) IP address (VIP) of the load balancer, a new VIP could be required for clients to connect to the new IKEv2 gateways.

As an alternative to using a load balancer, for clients that support the IKEv2 redirect feature, FlexVPN load balancer can be used. In a scenario such as this, clients will use either IKEv1 or IKEv2. It is uncommon for clients to support both protocols. Cisco AnyConnect has only supported IKEv2 since version 3. For this reason, clients can be migrated to the new VPN gateway using the IKEv2 redirect feature, and when all clients have been migrated, the previous IKEv1 gateway and load balancer may be decommissioned.

> **Note**
>
> If clients or peers connect to a VPN gateway using a Fully Qualified Domain Name (FQDN) and DNS is used, an additional DNS record will be required if an IKEv1 and IKEv2 gateway is used simultaneously during migration.

Asymmetric Routing

In situations where states of network flows are maintained and used by network or security devices, you must ensure that symmetrical flows occur. Figure 16-1 illustrates a typical scenario in which separate hardware is used for IKEv1 and IKEv2 protected traffic. The two routers at the same branch have a software firewall, such a zone-based firewall (ZBF), which inspects unencrypted (plaintext) traffic. If traffic is sourced from the LAN behind the branch routers, goes to Hub1 via the IKEv2-protected IPsec Security Association, and returns via Hub2 and the IKEv1-protected IPsec Security Association, the software firewall on Brach router2 will drop this traffic because no state exists.

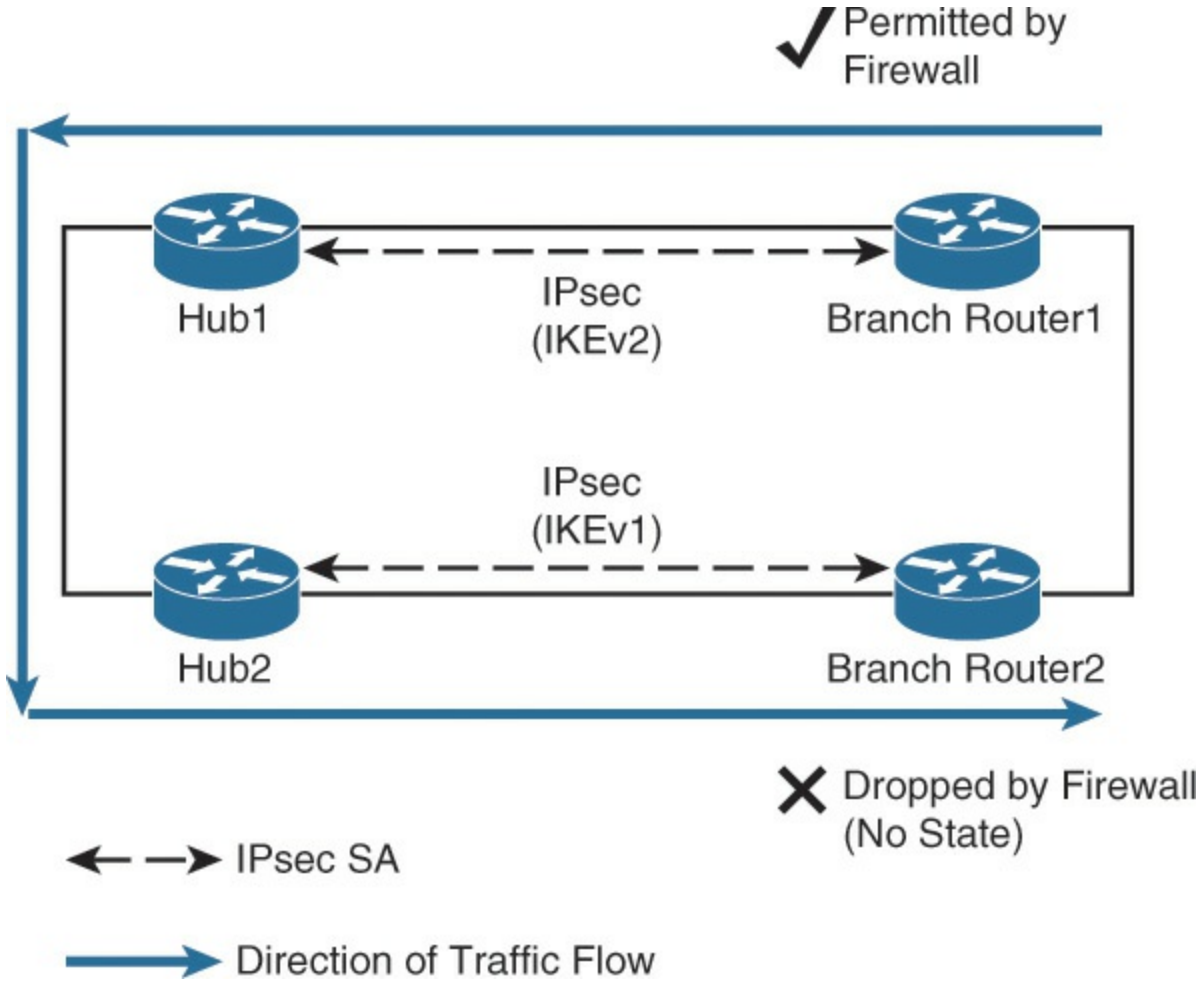


Figure 16-1 *Asymmetric Routing*

Example of other devices that maintain state are load balancers, proxies, Intrusion Prevention Systems (IPS), and Network Address Translation (NAT) devices. Care should be taken when migrating to ensure that these devices do not cause traffic to be dropped or blocked due to asymmetric routing.

Migration Strategies

In a transition from IKEv1 to IKEv2, there are a number of methods for migration that will depend on the type of topology being deployed and the equipment currently in use.

We can break the migration strategies down into two high level categories:

1. Ones where new equipment is required because of hardware, capacity, configuration, or licensing limitations, so the migration will involve swapping out existing hardware for new. This is known as a hard migration, in which there is a service outage when migrating between hardware.

2. Ones where the current equipment supports both IKEv1 and IKEv2 and both protocols will be operated at the same time. This is known as a soft migration, in which the amount of downtime is minimized and a slower approach can be taken.

Either method should be carefully planned to ensure that disruption is minimized.

Hard Migration

When new equipment is required, devices are swapped out or configurations removed and re-applied; testing should then occur within a lab environment to ensure that the new configuration will operate as intended.

Hard migrations will usually have a planned outage during which equipment will be swapped. Ensure that the ARP cache of adjacent devices is cleared when the new hardware is installed and ensure that management plane access to the new device is permitted.

For a hub-and-spoke topology, if the client is to be migrated in a hard migration, the hub site must be able to support both IKEv1 and IKEv2 clients. Having the hub support both types of clients or installing a new hub that will service IKEv2 clients can achieve this. Installing a new hub is a preferred option, for it allows the older hardware to be decommissioned once the migration has occurred.

If the hub is to service both IKEv1 and IKEv2 clients, then the cleanest method to separate access is to have a separate IP address for the IKEv1 tunnel source interface and another separate IP address for the IKEv2 tunnel source interface. This removes the chance of misconfigurations and allows for a clean differentiator between IKEv1 and IKEv2.

If architectures have an IPsec Security Association protecting traffic between two local-area networks (LAN) and both IKE peers are required to be swapped at the same time, the outage window should be scheduled accordingly. To enable ease of troubleshooting, security controls such as firewalls should be configured to allow ICMP echo (type 8 code 0) and echo-reply (type 0 code 0) from the tunnel source and destination addresses.

The configuration used when performing a hard migration depends on the implementation. From a new client device perspective, the configuration will be as you would configure any IKEv2 device, as explained in earlier chapters. For the hub device, the configuration will probably use virtual-templates; however, the exact configuration depends on the specific requirements.

If for some reason the new deployment was not successful, then reverting to the previous IKEv1 configuration should be a case of reinstalling the original hardware or reapplying the original configuration. A diagnosis can then be performed to determine the cause of the failure of the migration.

Soft Migration

When a soft migration is performed, both IKEv1 and IKEv2 will coexist on the same device for a period of time. For the time that both IKEv1 and IKEv2 are operational, a method must be used to differentiate sending traffic via an IPsec Security Association established using IKEv1 or IKEv2. This task is usually performed by the routing protocol, where traffic is continually sent down the IPsec Security Association created by IKEv1, until the IKEv2 SA and corresponding IPsec Security Association has been verified. Verification can be as simple as redistributing a small range of prefixes over the IKEv2-created IPsec Security Association and then sending traffic between peers who are consuming these prefixes.

Figure 16-2 illustrates the concept of testing an IKEv2-generated IPsec Security Association while sending the majority of traffic over the existing IKEv1-created IPsec Security Association. This can be achieved by setting a static route with the next hop of the tunnel interface using the command **ip route 192.168.10.1 255.255.255.255 tunnel 10**, with tunnel 10 being the tunnel protected by IKEv2.

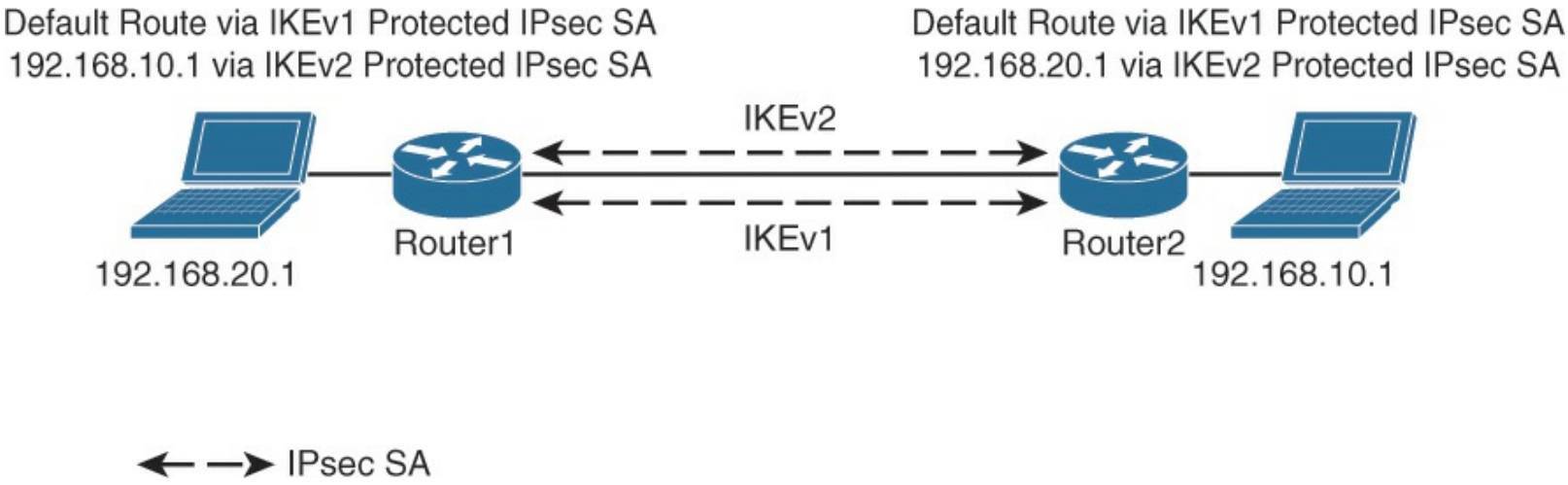


Figure 16-2 *Testing IKEv2-Protected IPsec Security Association*

Soft Migration Example

The following scenario provides guidance on migrating from an IKEv1 deployment to an IKEv2 deployment with minimal downtime and disruption.

This scenario will use IKEv2 routing as the new method to exchange prefixes between peers. For more complex architectures where a routing protocol is used, an adjustment should be made to ensure that the routing protocol used by IKEv2 does not override the routing protocol used by IKEv1 until the intended time.

We start with an initial configuration where IKEv1 is used to establish a secure connection between peers; this is a simple IPsec Security Association. Over the IPsec Security Association, EIGRP is used to create a neighbor relationship and distribute prefixes between devices.

The topology is illustrated in Figure 16-3. There are two routers currently connected with IKEv1.

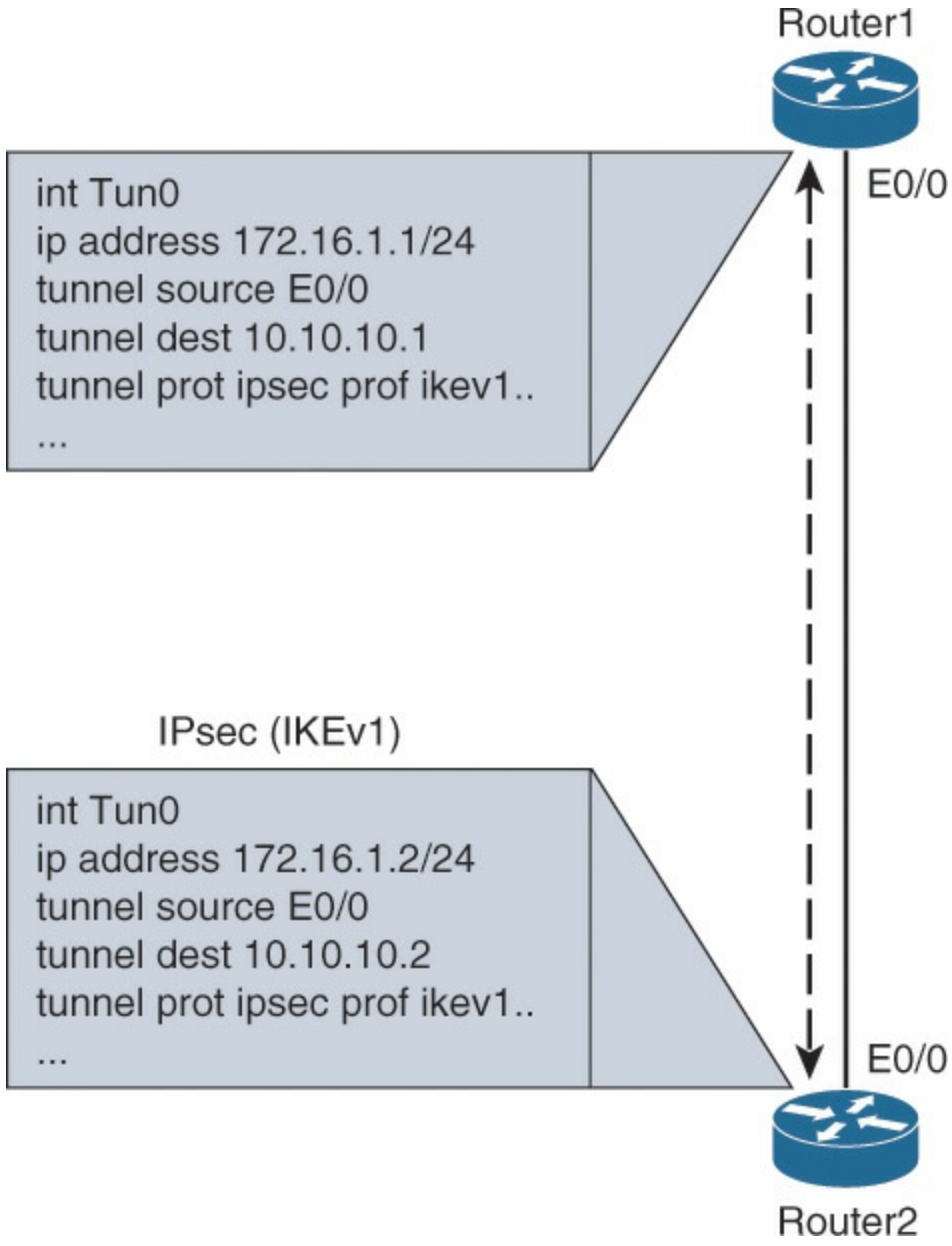


Figure 16-3 *IKEv1-Protected IPsec Security Association*

The following was created to simulate an already established IKEv1 deployment.

1. An IKEv1 policy is created; this is vaguely equivalent to a IKEv2 proposal.

```
crypto isakmp policy 10
 authentication pre-share
```

2. For simplicity, a pre-shared key is used.

Click here to view code image

```
crypto isakmp key cisco address 10.10.10.2
```

3. An IPsec profile is used, which will use the default IKEv1 policy created earlier.

```
crypto ipsec profile ikev1
```

4. A loopback interface is used that will allow traffic that will transverse the VPN to be sourced from and destined to.

Click here to view code image

```
interface Loopback0
```

```
 ip address 192.168.10.1 255.255.255.0
```

5. The tunnel interface is created with the relevant source interface configured and the destination address of Router2. This is protected by the IPsec profile created previously.

Click here to view code image

```
interface Tunnel0
 ip address 172.16.1.1 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.2
 tunnel protection ipsec profile ikev1
```

6. The interface used as the tunnel source.

Click here to view code image

```
interface Ethernet0/0
 ip address 10.10.10.1 255.255.255.0
```

7. EIGRP is used to establish a peer relationship over the tunnel interface and distribute the loopback prefix.

```
router eigrp 1
 network 172.16.1.0 0.0.0.255
 network 192.168.10.0
```

Router 2 has a similar configuration; the relevant details are displayed in the following example.

Click here to view code image

```
crypto isakmp policy 10
 authentication pre-share
crypto isakmp key cisco address 10.10.10.1

crypto ipsec profile ikev1

interface Loopback0
 ip address 192.168.20.1 255.255.255.0

interface Tunnel0
 ip address 172.16.1.2 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.1
 tunnel protection ipsec profile ikev1

interface Ethernet0/0
 ip address 10.10.10.2 255.255.255.0

router eigrp 1
 network 172.16.1.0 0.0.0.255
 network 192.168.20.0
```

The IKEv1 SA is created, which in turn creates the IPsec Security Association that is used to protect the tunnel interface. The following example illustrates the verification of the IKEv1 SA and the neighbor establishment over the corresponding tunnel interface.

The IKEv1 SA can be seen. Note that the state QM_IDLE indicates that this was successfully created.

Click here to view code image

```
Router1#show crypto isakmp sa
IPv4 Crypto ISAKMP SA
dst             src             state          conn-id status
10.10.10.2      10.10.10.1      QM_IDLE           1003 ACTIVE

IPv6 Crypto ISAKMP SA
```

The prefix for the IP address on the loopback interface on Router2 is distributed over EIGRP. Router1 will use the tunnel interface as the next hop to reach this prefix.

Click here to view code image

```
Router1#show ip route 192.168.20.0
Routing entry for 192.168.20.0/24
  Known via "eigrp 1", distance 90, metric 27008000, type internal
  Redistributing via eigrp 1
  Last update from 172.16.1.2 on Tunnel0, 00:25:33 ago
  Routing Descriptor Blocks:
  * 172.16.1.2, from 172.16.1.2, 00:25:33 ago, via Tunnel0
      Route metric is 27008000, traffic share count is 1
      Total delay is 55000 microseconds, minimum bandwidth is 100 Kbit
      Reliability 255/255, minimum MTU 1476 bytes
      Loading 1/255, Hops 1
```

Both routers will now be configured to use a second tunnel interface. This interface will be protected by an IPsec profile that uses IKEv2. Rather than using EIGRP as the mechanism to distribute prefixes between peers, IKEv2 routing will be used. As EIGRP uses a default Administrative Distance (AD) of 90 and IKEv2 routing will install prefixes in the RIB as static routes with a local AD of 1, the prefixes installed for IKEv2 routing will be amended to a value of 100, being less preferred than the EIGRP prefixes. This ensures that the RIB will always prefer the prefixes learned over EIGRP via the IKEv1-protected tunnel interface rather than the prefixes learned via the IKEv2-protected tunnel interface. If a large number of prefixes were required or these changed frequently, a routing protocol would be a better choice rather than IKEv2 routing. The following example shows the configuration required.

Note that in this configuration pre-shared key authentication is used with a weak shared-secret. For production environments, digital signatures should be used or a shared-secret containing sufficient entropy. For maximum security, next-generation encryption (NGE) algorithms should be used.

1. An access control list is created which contains the prefix that is to be sent to the peer router. This is the same prefix as Router1 distributes to Router2 using EIGRP over the IKEv1 IPsec Security Association.

Click here to view code image

```
access-list 50 permit 192.168.10.0 0.0.0.255
```

2. The default IKEv2 authorization policy is amended to send the prefixes created in access-list 50. Any prefixes received over IKEv2 will be applied with an administrative distance of 100.

Click here to view code image

```
crypto ikev2 authorization policy default
 route set interface
 route set access-list 50
 route accept any distance 100
```

3. An IKEv2 profile is created that is used for connectivity to Router2. As IKEv2 routing is used, DPD is enabled to ensure that the IKEv2 SA is torn down and corresponding prefixes are removed if connectivity is lost. To enable sending and receiving prefixes to/from the peer using IKEv2 routing, AAA authorization is enabled.

Click here to view code image

```
crypto ikev2 profile ikev2
 match identity remote address 10.10.10.2 255.255.255.255
 authentication remote pre-share key cisco123
 authentication local pre-share key cisco123
 dpd 30 10 on-demand
 aaa authorization group psk list default default
```

4. An IPsec profile is created that uses the IKEv2 profile created previously.

```
crypto ipsec profile ikev2
 set ikev2-profile ikev2
```

5. The tunnel interface is created with the relevant source interface configured and the destination address of Router2. This is protected by the IPsec profile created previously, enabling IKEv2 on this interface. To ensure that the correct IKEv2 profile is selected and not the IKEv1 profile, this is defined using the **ikev2-profile** command.

Click here to view code image

```
interface Tunnel10
 ip address 172.16.2.1 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.2
 tunnel protection ipsec profile ikev2 ikev2-profile ikev2
```

The configuration for Router2 is very similar. The following example illustrates the configuration on Router2 with the differences being highlighted.

Click here to view code image

```
access-list 50 permit 192.168.20.0 0.0.0.255

crypto ikev2 authorization policy default
 route set interface
 route set access-list 50
 route accept any distance 100

crypto ikev2 profile ikev2
 match identity remote address 10.10.10.1 255.255.255.255
 authentication remote pre-share key cisco123
 authentication local pre-share key cisco123
 dpd 30 10 on-demand
 aaa authorization group psk list default default

crypto ipsec profile ikev2
 set ikev2-profile ikev2

interface Tunnel10
 ip address 172.16.2.2 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.1
```

```
tunnel protection ipsec profile ikev2 ikev2-profile ikev2
```

Each router now has two tunnel interfaces created between each other. One is protected using IKEv1 and the other using IKEv2. The following example shows the commands used to verify the new tunnel interface and the protection applied.

The IKEv2 SA includes the remote prefix that was sent over IKEv2 routing.

Click here to view code image

```
Router1#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         10.10.10.1/500        10.10.10.2/500        none/none            READY
      Encr: AES-CBC, keysize: 256, PRF: SHA512, Hash: SHA512, DH Grp:5, Auth sign:
PSK, Auth verify: PSK
      Life/Active Time: 86400/629 sec
      CE id: 1004, Session-id: 2
      Status Description: Negotiation done
      Local spi: CDB7428C1D03F3AB       Remote spi: F5C4546346F689A9
      Local id: 10.10.10.1
      Remote id: 10.10.10.2
      Local req msg id:  3              Remote req msg id:  0
      Local next msg id: 3              Remote next msg id: 0
      Local req queued:  3              Remote req queued:  0
      Local window:      5              Remote window:      5
      DPD configured for 30 seconds, retry 10
      Fragmentation not configured.
      Extended Authentication not configured.
      NAT-T is not detected
      Cisco Trust Security SGT is disabled
      Initiator of SA : Yes
      Remote subnets:
      192.168.20.0 255.255.255.0
```

Although the prefix for 192.168.20.0/24 was received, it will have an AD of 100. This is not installed into the RIB as the prefix is already populated by the prefix installed by EIGRP, with an AD of 90.

Click here to view code image

```
Router1#show ip route 192.168.20.0
Routing entry for 192.168.20.0/24
  Known via "eigrp 1", distance 90, metric 27008000, type internal
  Redistributing via eigrp 1
  Last update from 172.16.1.2 on Tunnel0, 00:11:56 ago
  Routing Descriptor Blocks:
  * 172.16.1.2, from 172.16.1.2, 00:11:56 ago, via Tunnel0
      Route metric is 27008000, traffic share count is 1
      Total delay is 55000 microseconds, minimum bandwidth is 100 Kbit
      Reliability 255/255, minimum MTU 1476 bytes
      Loading 1/255, Hops 1
```

Figure 16-4 illustrates both the IKEv1 and IKEv2 tunnels coexisting. Dataplane traffic will flow via the IKEv1-protected tunnel interface.

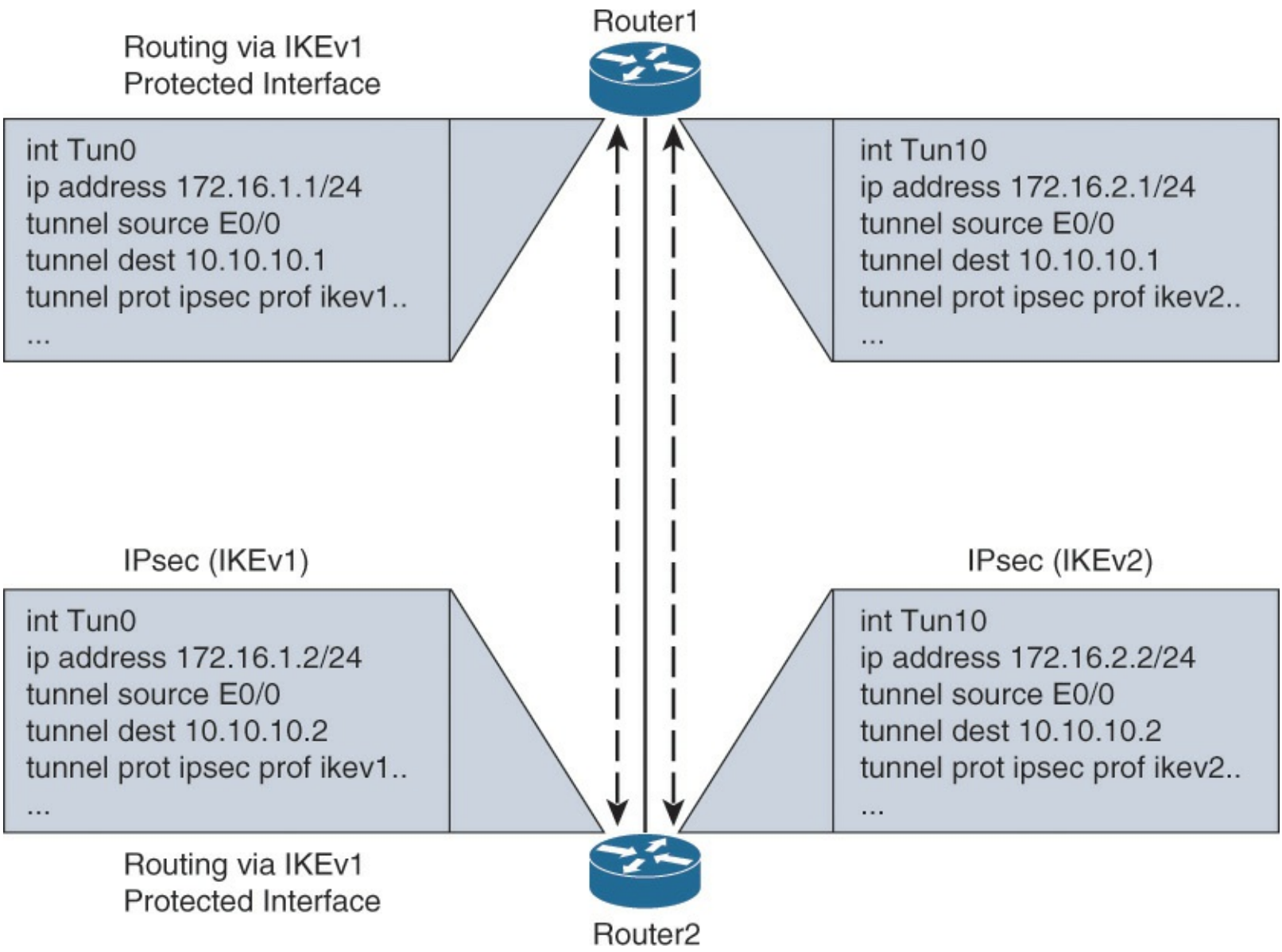


Figure 16-4 *IKEv1 and IKEv2 Tunnels*

When the IKEv1-protected tunnel interface (Tunnel 0) is shut down, the EIGRP neighborship is deleted, and the prefix for 192.168.20.0/24 that was learned via EIGRP is removed. At this point the prefix that was received via the IKEv2-protected tunnel interface (Tunnel 10) is installed into the RIB. The following example illustrates this behaviour, where debugging of IP routing was enabled.

Click here to view code image

```
Router1(config)#interface tunnel 0
Router1(config-if)#shutdown
is_up: Tunnel0 0 state: 6 sub state: 1 line: 0
 RT: interface Tunnel0 removed from routing table
 RT: del 172.16.1.0 via 0.0.0.0, connected metric [0/0]
 RT: delete subnet route to 172.16.1.0/24
 RT: del 172.16.1.1 via 0.0.0.0, connected metric [0/0]
 RT: delete subnet route to 172.16.1.1/32
 %DUAL-5-NBRCHANGE: EIGRP-IPv4 1: Neighbor 172.16.1.2 (Tunnel0) is down: interface
down
 RT: delete route to 192.168.20.0 via 172.16.1.2, eigrp metric [90/27008000]
 RT: no routes to 192.168.20.0, delayed flush
 RT: delete network route to 192.168.20.0/24
 RT: updating static 192.168.20.0/24 (0x0)   :
    via 0.0.0.0 Tu10   0 1048578
 RT: add 192.168.20.0/24 via 0.0.0.0, static metric [100/0]
```

The prefix is now accessible via the tunnel interface (Tunnel 10) protected by IKEv2.

Click here to view code image

```
Router1#show ip route 192.168.20.0
Routing entry for 192.168.20.0/24
  Known via "static", distance 100, metric 0 (connected)
  Tag 1
  Routing Descriptor Blocks:
  * directly connected, via Tunnel10, permanent
      Route metric is 0, traffic share count is 1
      Route tag 1
```

Now all traffic that is destined to the 192.168.20.0/24 prefix will flow over the tunnel that was created using IKEv2. This can be seen in Figure 16-5; as soon as the IKEv1-protected tunnel interface is shut down, traffic will flow over the IKEv2 tunnel interface.

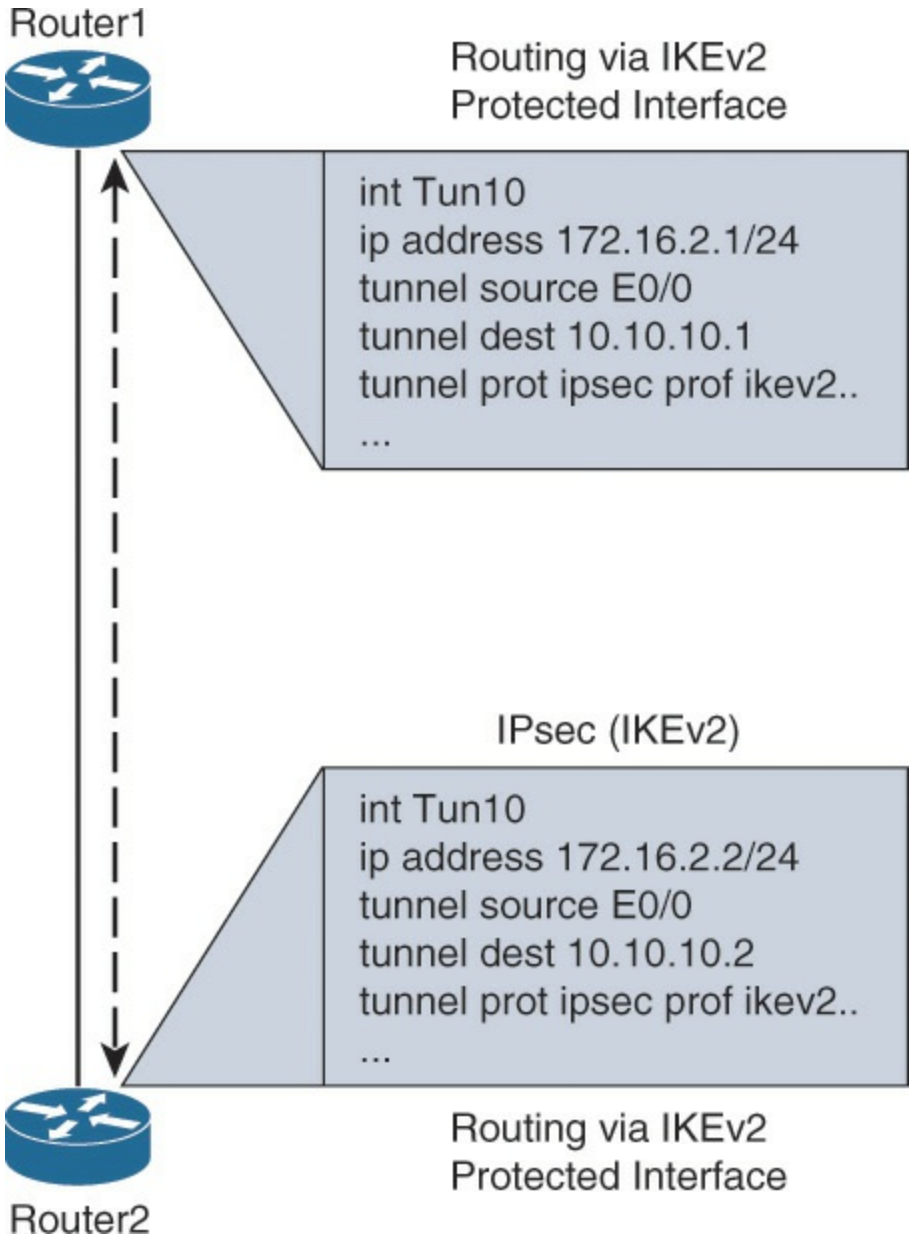


Figure 16-5 *IKEv2-Protected Tunnel Interface*

For migration scenarios, it is advised that the IKEv2 SA is left to rekey a number of times to ensure that there is stability before bringing down the tunnel created by IKEv1. This check can verify that IKEv2 is functioning correctly.

Once it has been determined that the IKEv2 tunnel is working as intended, the IKEv1 tunnel interface

can be shut down. When it is certain that the IKEv1 configuration is no longer required, this can be removed.

In the previous examples, the IPsec profile contained only IKEv1 or IKEv2 profiles, never mixed. It is currently impossible to specify an IKEv1 and an IKEv2 profile within an IPsec profile.

An IPsec profile will only allow either an IKEv1 profile or an IKEv2 profile to be attached. It is not possible to have both an IKEv1 and an IKEv2 profile configured within an IPsec profile.

The following example illustrates an attempt to apply both an IKEv1 and a IKEv2 profile to an IPsec profile and the error message generated.

Click here to view code image

```
Router(config)#crypto ipsec profile ipsec
Router(ipsec-profile)#set isakmp-profile ikev1
Router(ipsec-profile)#set ikev2-profile ikev2
% WARNING: Configuring Ikev2 profile will remove the Ikev1 profile configured
under this ipsec profile!
```

When tunnel protection is applied, the specific IKEv1 or IKEv2 profile that will be used can be configured. The following example illustrates applying an IPsec profile that is configured to use an IKEv1 profile; however, because the IKEv2 profile is specified in the **tunnel protection** command, the IKEv2 profile will be used instead, and IKEv1 traffic will not be processed for this tunnel interface.

Click here to view code image

```
Router(config)#interface tunnel 1
Router(config-if)#tunnel protection ipsec profile ipsec_profile ?
  ikev2-profile   Specify ikev2 profile for the crypto connection.
  isakmp-profile  Specify isakmp profile for the crypto connection.
  shared          Use a shared socket for the crypto connection.

Router(config-if)#tunnel protection ipsec profile ipsec_profile ikev2-profile
ikev2
```

Although this is not a common deployment and similar goals can be achieved using a configuration where a separate tunnel protection is applied per tunnel interface, this can be used for migrations from IKEv1 to IKEv2.

Migration Verification

When IKEv2 is active for the first time, basic checks should be performed to verify that the IKE and IPsec Security Association have been created as intended. The **show crypto ikev2 sa** command can be used to verify the status of the IKEv2 SA, with the **detail** keyword adding more verbose output. The IPsec Security Association can be verified along with the IKEv2 SA with the **show crypto session** command.

The IKEv2 SA should be left to rekey a number of times to ensure that rekey is successful; if PFS is enabled, this can be validated also.

The following example illustrates how to determine if PFS is enabled. Note that PFS will only show once the first CHILD_SA has been created, so you will need to allow the IPsec Security Association to rekey, which is by default 3600 seconds.

Click here to view code image

```
Router#show crypto ipsec sa

interface: Virtual-Access1
    Crypto-map tag: Virtual-Access1-head-0, local addr 10.10.10.1

   protected vrf: (none)
   local  ident (addr/mask/prot/port): (10.10.10.1/255.255.255.255/47/0)
   remote ident (addr/mask/prot/port): (10.10.10.2/255.255.255.255/47/0)
   current_peer 10.10.10.2 port 500
     PERMIT, flags={origin_is_acl,}
    #pkts encaps: 726, #pkts encrypt: 726, #pkts digest: 726
    #pkts decaps: 1521, #pkts decrypt: 1521, #pkts verify: 1521
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0
    #pkts not decompressed: 0, #pkts decompress failed: 0
    #send errors 0, #recv errors 0

     local crypto endpt.: 10.10.10.1, remote crypto endpt.: 10.10.10.2
     plaintext mtu 1458, path mtu 1500, ip mtu 1500, ip mtu idb Ethernet0/0
     current outbound spi: 0x3F82A9C5(1065527749)
     PFS (Y/N): Y, DH group: group19
```

If PKI is used with certificate revocation, checks should occur to ensure that revocation is working correctly and that when a client's certificate is revoked, access for that client is prevented due to a failed authentication.

Assuming that a like-for-like migration has occurred, the same number of peer devices should exist once the configuration has moved from IKEv1 to IKEv2. The following example illustrates viewing the number of active IKEv2 SAs and then viewing deeper details about these with the **show crypto session brief** and the **show crypto IPsec sa** commands. Note that this **show** command does not differentiate between IKEv1 and IKEv2 sessions.

Click here to view code image

```
Router#show crypto ikev2 stats
------------------------------------------------------------------------------
                          Crypto IKEv2 SA Statistics
------------------------------------------------------------------------------
System Resource Limit:   0        Max IKEv2 SAs: 0        Max in nego(in/out):
40/400
Total incoming IKEv2 SA Count:    2        active:        2        negotiating: 0
Total outgoing IKEv2 SA Count:    0        active:        0        negotiating: 0
...

Router#show crypto session brief
Status: A- Active, U - Up, D - Down, I - Idle, S - Standby, N - Negotiating
        K - No IKE
ivrf = (none)
           Peer     I/F        Username          Group/Phase1_id   Uptime Status
     10.10.10.2     Vi1                  hostname=R2.cisco.com,cn 01:30:00    UA
     10.10.10.1     Vi2                  hostname=R1.cisco.com,cn 00:07:00    UA
```

In most cases the same prefixes will exist in the RIB once migration has occurred, verifying that the RIB is populated as intended can be performed using the **show ip route** command.

Consideration for Topologies

The following topologies are examined, and the main considerations that pertain to each are described.

Common IPsec VPN topologies can be broken down into the following categories:

Site-to-Site

In this setup, a tunnel is created between two routers. This is a common method to securely connect two organizations or data centers together. Normally a single tunnel interface is used on each device. Having just a single pair of IPsec Security Associations allows for a simple migration path, in which a new tunnel can be created and brought up alongside the existing tunnel.

Should crypto maps be currently used, then there is the possibility that tunnel interfaces can be used instead. Should this occur, additional configuration may be required to ensure that only specific traffic is permitted to transverse the new tunnel interface.

If the current equipment supports IKEv2, then a new pair of tunnel interfaces can be created and migrated from IKEv1 to IKEv2. This is the simple approach and allows reuse of hardware; however, hardware capacity needs to be taken into account. For this scenario, routing protocols could be used over the existing IPsec Security Association. Thought should go into how the routing is going to be migrated from the existing architecture to the new design when traffic protected using IKEv2 will become active.

The following example illustrates moving a site-to-site VPN from using IKEv1 to IKEv2. High-level steps are shown to illustrate the concepts. For specific configuration information, please consult the previous chapters. This merely illustrates a method to follow for a migration with minimal disruption. This methodology can be implemented only on devices that are IKEv2-capable.

Figure 16-6 illustrates a simple site-to-site VPN between two routers. Currently this is using IKEv1.

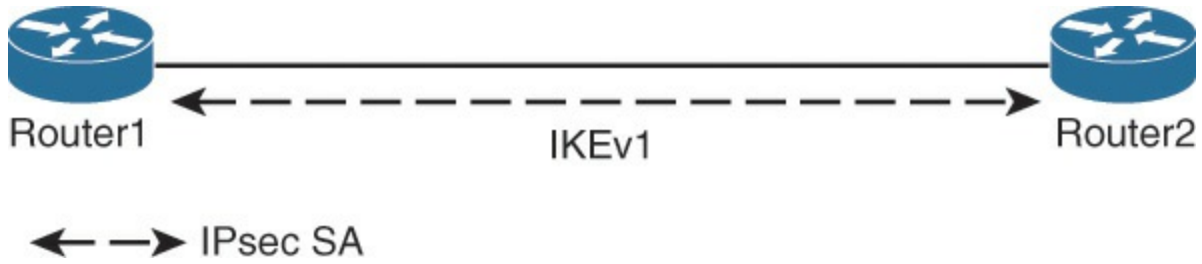


Figure 16-6 *Site-to-Site VPN Using IKEv1*

Figure 16-7 shows a second protected tunnel interface created on both routers. This second IPsec protected tunnel is created using IKEv2. Routing protocols should be configured so that the IKEv1-protected VPN has preference over the IKEv2-protected VPN. Routing via the second tunnel must also be verified, which can be achieved manually, depending on routing protocol or static IKEv2 routing.

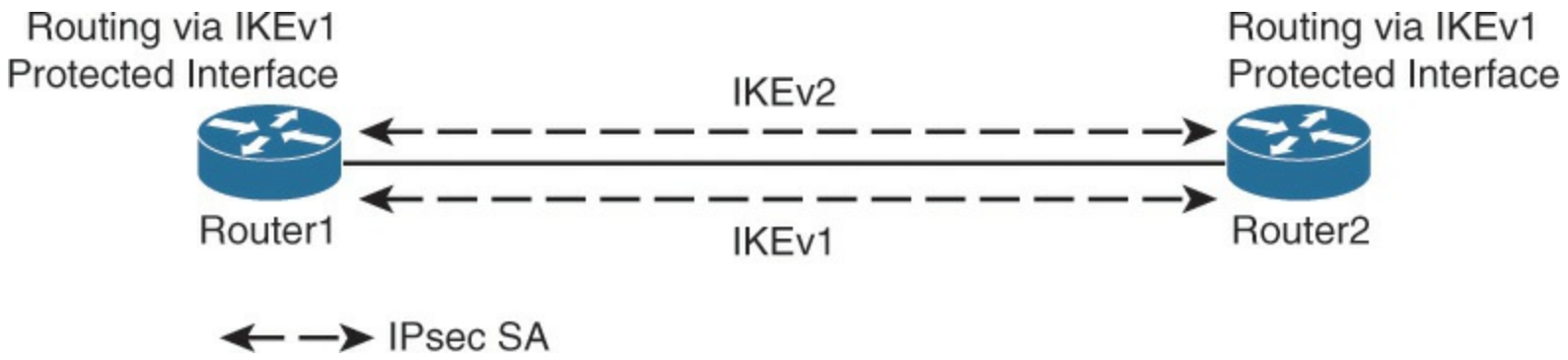


Figure 16-7 *Site-to-Site VPN with IKEv1 and IKEv2*

In the following figure there are multiple tunnels on each peer. This simulates both peers running IKEv1 and IKEv2 simultaneously. It could be the case that the IKEv1 tunnel is destined to one peer and the IKEv2 to another; however, for simplicity this is not shown, but the concept would be the same.

Figure 16-8 illustrates the IKEv1-protected VPN being shut down. Traffic will now flow over the VPN protected by IKEv2. Once it has been verified that the new IKEv2-protected VPN is functioning correctly, the previous configuration can be removed to minimize confusion.

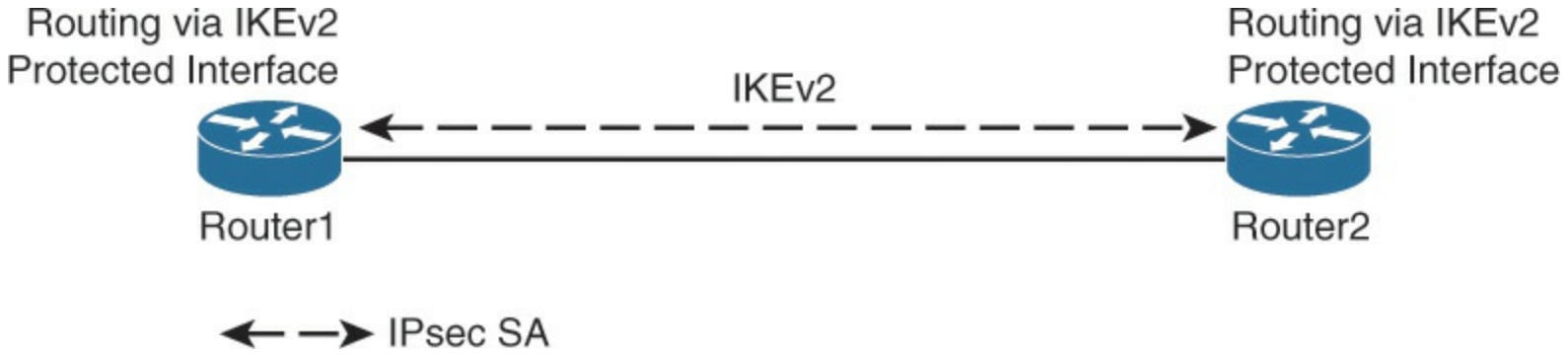


Figure 16-8 *Site-to-Site VPN with IKEv2*

Hub and Spoke

Hub-and-spoke topology is used when remote devices connect to a single location. An example is small offices or home workers who connect to a central location. It is common for EzVPN to be used in this scenario. Remote sites will usually have a default route or summary route via the hub. In situations where spoke devices have dual connections to hubs, IKEv2 routing is very suitable, for different paths can have different local administrative distances.

There are many features of FlexVPN that were not possible using previous VPN technologies. Features such as backup peers and/or tunnel based activation are well suited to hub-and-spoke architectures and can bring a number of business benefits that were previously unavailable.

For more complex routing scenarios, a routing protocol can be used over a tunnel interface, but care should be taken for lower-end models to ensure that the capacity is adequate if multiple tunnels and routing protocols are required for a transition phase.

If different groups of spoke devices have different requirements for policies such as QoS or ACLs, then the method to apply these must be defined. If a name-mangler is going to be used to extract a property within the IKE identity, it is critical that the property extracted is correctly defined and operates as intended.

The following example shows how a hub-and-spoke deployment can be migrated from a hub that is

running IKEv1 to a new hub that is running IKEv2. As the spoke device is going to be running IKEv1 and IKEv2 concurrently, this would be classified as a soft move.

Figure 16-9 illustrates the topology. A spoke router has an IKEv1 connection to a hub router (Hub1). Hub1 cannot support IKEv2, hence another hub (Hub2) is provisioned. The spoke can support both IKEv1 and IKEv2.

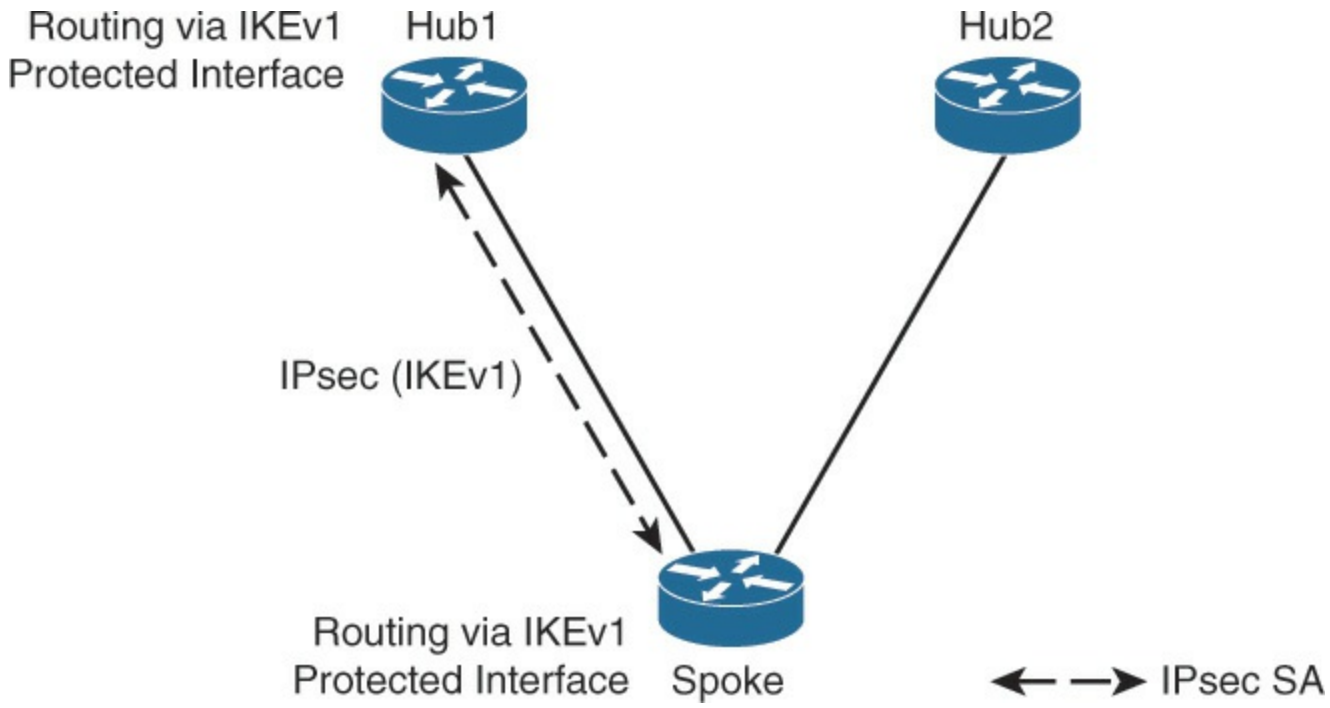


Figure 16-9 *Spoke to Hub using IKEv1*

Figure 16-10 illustrates a second protected tunnel interface created on the spoke to Hub2. The spoke now has two active tunnel interfaces, one to Hub1 protected by IKEv1 and one to Hub2 that is protected by IKEv2. Routing protocols should be configured so the IKEv1-protected VPN has preference over the IKEv2-protected VPN. Routing via the second tunnel must also be verified.

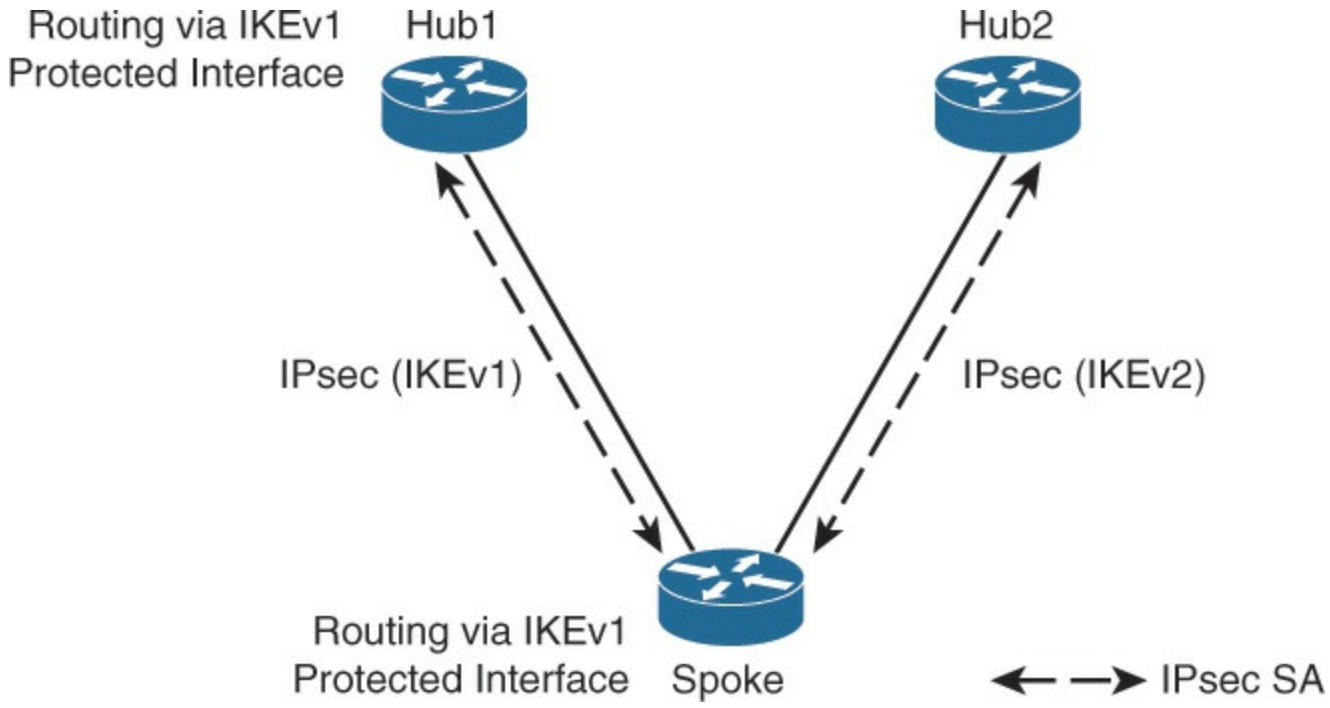


Figure 16-10 *Spoke to Hub1 Using IKEv1 and Hub2 Using IKEv2*

Figure 16-11 illustrates the IKEv1 tunnel interface being shut down on the spoke. Traffic will then flow via the IKEv2-protected interface to Hub2. Once it has been verified that the new IKEv2

protected VPN is correctly functioning the previous configuration can be removed to minimize confusion.

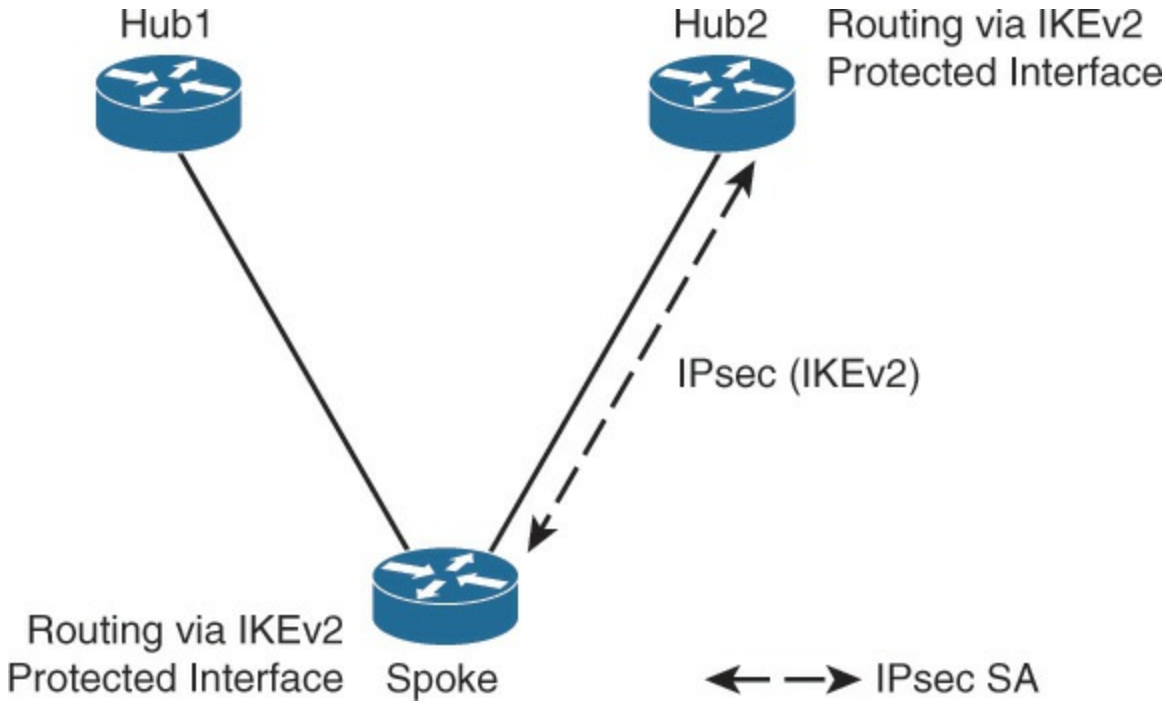


Figure 16-11 *Spoke to Hub2 Using IKEv2*

Once all spokes have been migrated from Hub1 to Hub2, Hub1 can be decommissioned and removed from service.

In the example shown above, if the Hub did support both IKEv1 and IKEv2 and there was sufficient capacity, both tunnels could be configured on the single hub.

If the spoke hardware could not support IKEv2, the same scenario could be achieved by physically swapping the spoke device for a hardware model that supported IKEv2. In this case a hard move would occur.

Remote Access

Remote access architectures allow for a number of clients to connect to a central location. The clients are usually software clients, small-form-factor routers, or similar devices. Internet of Things (IoT) is now an example of what could be described as remote access. Clients will be allocated an address dynamically and will usually tunnel all traffic to the VPN gateway. In some situations traffic might be split-tunneled, where the IPsec Security Association will not protect all traffic, certain traffic will be sent outside of the IPsec Security Association.

When new software clients are required, an upgrade of the operating system could be needed, or additional software patches be applied to clients. When IKEv2 is used, mandatory attributes might be required by the client that were not required in IKEv1.

In many instances remote access clients might require IKEv2 fragmentation to be enabled, especially if the authentication method has changed to using a certificate-based architecture. In this case, IKEv2 fragmentation should be considered and enabled if required.

Care should be taken in designing the allocation of client IP addresses, the pool must be re-distributed into the routing protocol to permit return traffic when accessing resources on the internal network. Clients should have full access to the resource that is needed, if multiple types of users or groups require separated access this will need to be integrated into the solution. It could be that multiple

IVRFs are used for each group of users, or that a simple access control list (ACL) is applied to restrict or permit access.

Figure 16-12 illustrates a typical migration where two VPN gateways are being used to serve remote clients. The IP address pool that is used to allocate clients IP address must be redistributed internally to allow a return path for clients' traffic. A common mistake when migrating is not redistributing this subnet into the local routing protocol, which results in clients having unidirectional access to resources on the protected LAN.

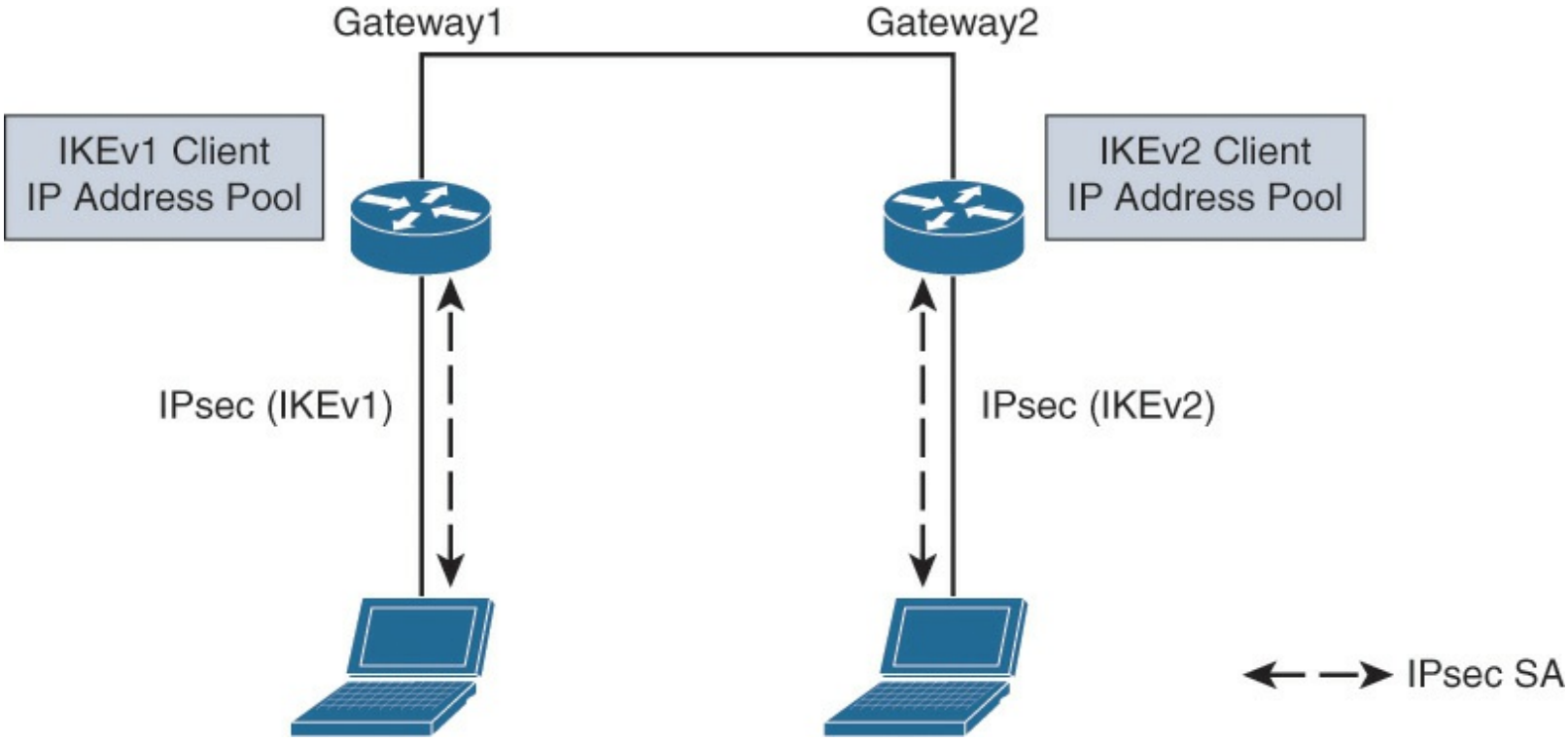


Figure 16-12 *Remote Access VPN*

QoS can be enabled easily on remote access VPN architectures when using dVTI. This might not have been possible in the previous deployment and care should be taken to ensure that traffic does not exceed set rates and is rate limited or dropped.

If single-factor authentication is needed or dual-factor using aggregate authentication, then AnyConnect will be needed.

In many remote access scenarios in which users are connecting to a VPN gateway running IKEv2, new hardware will be required. The rationale behind this statement is that many remote access deployments are still running IKEv1 or TLS and at the time of this writing are moving to using IKEv2.

Summary

When migrating from IKEv1 to IKEv2, there is no one-size-fits-all model. Just as every IP network is unique, every migration will be unique. For this reason all considerations covered within this chapter potentially need to be addressed.

It is necessary to be aware of the considerations when moving to IKEv2, such as what features are supported by current software and hardware, to ensure a smooth transition. Knowing the capacity of the current system is crucial. Deploying a migration that could cause disruption due to exceeding capacity could potentially be catastrophic.

When remote access solutions are migrated, a number of features provided by FlexVPN can be used

to ensure a smooth transition. IKEv2 routing can be used to easily amend locally installed prefixes to install backup routes that become active once the legacy IKEv1 tunnel is shutdown.

There are a few key points to remember when migrating. Always leave the new IKEv2 tunnel running for a period of time to ensure that rekeying occurs. There are a number of restrictions when running IKEv1 and IKEv2 on the same device. Separate tunnel interfaces should be configured, protected by each protocol, and these should be clearly marked to ensure that the correct tunnel is shutdown when required.

Index

Symbols

+ (plus sign), 54
3DES, 111
3DS-CBC, 514

A

AAA (authentication, authorization, and accounting), 418
debug commands, 501
RADIUS-based AAA, authorization, 436
AAA accounting, FlexVPN server, 287
AAA authentication method, 278
AAA authorization, deriving virtual access configurations from (FlexVPN server), 293–294
aaa authorization, 223, 225, 229, 233
aaa authorization group, 229, 237–238
aaa authorization group override, 294
aaa authorization user, 285, 484
AAA infrastructure, 221–223
configuring for FlexVPN, 222–223
aaa new-model command, 222
AAA-based pre-shared keys
FlexVPN, 381–382
branch-1 router configuration, 382–383
branch-2 router configuration, 383
hub router configuration, 383–384
RADIUS server configuration, 384–386
FlexVPN server, 283–284
configuring, 284
examples, 285–287
RADIUS attributes, 285
acceptable algorithms, 111
access, remote access (migration strategies), 565–566
access control, 4
accounting, 418
FlexVPN server, 287–290
advanced features, FlexVPN client, 336
AES CBC mode, 112
AES GCM mode, 112

AES-CBC, 514
AES-CBC-128, 111, 513
AES-CBC-192, 111
AES-CBC-256, 111
AES-CCM, 513–514
AES-CMAC-96, 515
AES-CTR, 514
AES-GCM, 513–514
AES-GCM-128, 111
AES-GCM-256, 111
AES-GMAC1, 514
AES-GMAC2, 513
AES-XCBC-MAC-96, 515
Aggregate authentication, 315
Aggregate XML, 315
Aggregation Services Routers (ASR), 87
Aggressive Mode, 70
AH (Authentication Header), 2–3
anti-replay services, 16–17
overhead, 509–510
algorithms
combined algorithm overhead, 512–513
hash algorithms, for signatures, 163
IKEv2 encryption, Cisco IOS, 111
IKEv2 integrity, in Cisco IOS, 113
integrity algorithms, 511–512
PRF (pseudorandom function), 41
pseudorandom function, IKEv2, 115
AMx, 70
anti-denial of service, 72
anti-replay services, 4, 16–17
ESP (Encapsulating Security Payload), 18
IPsec, 491–494
AnyConnect
EKU (Extended Key Usage), 469
FlexVPN server
auto-reconnect configuration attributes, 310–311
IKEv2, 309–310
name verification, 468

XML configurations, 282–283
AnyConnect-EAP, 146, 315, 315–316
configuring dual-factor authentication, 324–325
dual-factor authentication, 320–324
user authentication, 316–318
configuring, 318–319
AnyConnect-EAP XML messages, dual-factor authentication, 322–324
APPLICATION_VERSION, 257
ASR (Aggregation Services Routers), 87
asymmetric cryptography, 8, 11–12
Diffie-Hellman exchange, 8–11
asymmetric keys, 132
asymmetric routing, considerations when moving to IKEv2, 547–548
attacks, MITM (man-in-the-middle) attack, 13
attributes
APPLICATION_VERSION, 257
backup-gateway attributes, configuring, 347
certificate attributes, 469
Cisco private use configuration attributes, 257–258
configuration attributes
FlexVPN, 253–258
IKEv2 auto-reconnect, 310–311
configuration payload, 59
default-domain attributes, configuring, 344
FlexVPN, 253
interface-config AAA attribute, 293–294
interface-config attribute, 296
INTERNAL_IP4_ADDRESS, 254
INTERNAL_IP4_DNS, 256
INTERNAL_IP4_NBNS, 257
INTERNAL_IP4_NETMASK, 254
INTERNAL_IP4_SUBNET, 255
INTERNAL_IP6_ADDRESS, 255
INTERNAL_IP6_DNS, 256
INTERNAL_IP6_SUBNET, 256
IPv4 DNS server attributes, configuring, 340
IPv4 WINS attribute, 343
IPv6 DNS server attributes, configuring, 341
RADIUS attributes
AAA-based pre-shared keys, 285

CoA (change-of-authorization), 303–304
FlexVPN server, 325–329
Split-DNS, configuring, 341
audit-session-id, 299, 303–304
AUTH payload, 13
authentication, 14, 73–74, 418
certificate-based authentication, 147–149
considerations when moving to IKEv2, 546–547
data origin authentication, 4
debugging, with PKI, 470
dual-factor authentication, AnyConnect-EAP, 320–324
EAP (Extensible Authentication Protocol), 434–436
EAP authentication. ***See*** EAP authentication, FlexVPN server
ECDSA (Elliptic-Curve Digital Signature Algorithm) authentication, 194–200
troubleshooting, 465–468
ESP (Encapsulating Security Payload), 17–18
IKE_AUTH, 42–43
IKEv2, 45
EAP (Extensible Authentication Protocol), 48–50
pre-shared-key-based authentication, 47
signature-based authentication, 46
IP Authentication Header, 15
PKI (Public Key Infrastructure), 431–434
pre-shared key authentication, 147
pre-shared keys, 478–479
pre-shared-key-based authentication, 47
PSK, 429
RSA authentication, troubleshooting, 465–468
RSA authentication using HTTP URL lookup, 200–207
signature-based authentication, 46
user authentication, AnyConnect-EAP, 315
authentication command, 147
Authentication Header (AH), 2–3, 15
authentication header overhead, 509–510
authentication methods, IKEv2 profiles, 145–149
authentication pre-shared keys, 429–431
authentication remote anyconnect-eap, 323
authorization, 418, 485–487
FlexVPN, 231–233
configuring, 233

group authorization, 237–241
implicit authorization, 242–245
user authorization, 235–237
RADIUS-based AAA, 436
authorization policy
default IKEv2 authorization policy, 229–231
IKEv2, 228–229
authorization types, FlexVPN, 245–250
auto detection of tunnel transport and encapsulation, 297–298
auto tunnel mode, 99
auto-detection of tunnel encapsulation and transport, FlexVPN, 219–221
automatic mode, tunnel initiation, FlexVPN client, 350
AutoReconnect, 311
auto-reconnect, FlexVPN server, 309–310
configuration attributes, 310–311
configuring, 313–315
smart DPD, 311–313
AutoReconnectBehavior, 311
avoid, 111

B

backup gateway lists, FlexVPN client, 347
backup gateways, FlexVPN client, 346
backup group command, 353
backup groups, FlexVPN client, 353–354
Backup Peers, 544
backup peers, FlexVPN hub resiliency, 411
backup-gateway attributes, configuring, 347
boolean expressions, tracking, lists of objects, 350–352
branch 1 configuration
FlexVPN, spoke configuration, 392–394
FlexVPN client, group and user authorization, 386
FlexVPN spoke, verification, 397–399
branch 2 configuration
FlexVPN, spoke configuration, 394–395
FlexVPN client, user authorization, 387
FlexVPN spoke, verification, 399–400
branch-1 router configuration, AAA-based pre-shared keys (FlexVPN), 382–383
branch-2 router configuration, AAA-based pre-shared keys (FlexVPN), 383
buffers, capture buffers, 457

building blocks of
FlexVPN
Cisco IOS AAA infrastructure, 221–223
IKEv2, 213–214
point-to-point tunnel interfaces, 214–221
FlexVPN client
FlexVPN client profiles, 334
IKEv2 configuration exchange, 334
NAT (Network Address Translation), 335
object tracking, 334
static P2P tunnel interfaces, 334
IPsec, 2
access control, 4
anti-replay services, 4
confidentiality, 4
connectionless integrity, 4
data origin authentication, 4
key management protocol, 3
SAs (Security Associations), 3
security protocols, 2–3
security services, 3
TFC (Traffic Flow Confidentiality), 4–5

C

CA (certificate authority), 12
PKI (Public Key Infrastructure), 160–162
CAC (Call Admission Control), IKEv2, 157
Cookie Challenge and Call Admission Control, 207–210
cached keyword, 234
capacity, considerations when moving to IKEv2, 542–543
capture buffers, 457
capture points, 458
CBC (cipher block chaining), 510
CDP (Cisco Discovery Protocol), 86
CDP (CRL Distribution Point), 475
CEF (Cisco Express Forwarding), 94
CERT, 148
certificate attributes, 469
certificate authority. *See* CA (certificate authority)
certificate authority (CA), PKI (Public Key Infrastructure), 160–162

certificate expiry, 470–472
session deletion, IKEv2, 184
certificate maps, matching, peers, 472–473
certificate requests, 148
IKEv2, 38–39
HTTP_CERT_LOOKUP_SUPPORTED, 39
certificate revocation, 473–476
Certificate Revocation Lists (CRLs), 163
certificate revocation method, 163
session deletion, 182
certificate-based authentication, 147–149
certificates
certificate expiry, 470–472
digital certificates, 12
HTTP URL-based certificate lookup, 156
matching peers, 140–141
certification revocation list (CRL), 181
CERTREQ, 148
CertReq, 26
CERTREQ payload, 476
CFG_ACK, 252
CFG_REPLY, 251
CFG_REQUEST, 251
CFG_SET, 252
change-of-authorization (CoA), 303–304
Child SAs, 24
childless initiation, 66
cipher block chaining (CBC), 510
Cisco AV pair, 325
Cisco Discovery Protocol (CDP), 86
Cisco Express Forwarding (CEF), 94
Cisco IKEv2 AnyConnect clients, 330
Cisco IOS
algorithms
IKEv2 encryption, 111
IKEv2 integrity, 113
Diffie-Hellman group, 114
IP sec configuration, 166–167
IPsec configuration

examples, 168
smart defaults, 168–169
IPsec profiles, 167
PKI (Public Key Infrastructure), 159–160
CA (certificate authority), 160–162
pseudorandom function algorithms, IKEv2, 115
Cisco IOS AAA infrastructure, 221–223
configuring for FlexVPN, 222–223
Cisco IPsec flow monitor MIB, 425
Cisco meta data (CMD), 179
Cisco private use configuration attributes, 257–258
Cisco unity attributes, 253
clamping, MSS (Maximum Segment Size), 526–527
clear crypto ikev2 diagnose error, 461
clear crypto ikev2 sa, 186
clear crypto ikev2 sa remote, 360
clear crypto session, 186
clearing, IKEv2 FlexVPN client sessions, 360
client awareness, considerations when moving to IKEv2, 545
client connect tunnel interface-number, 348
client connect Tunnelo, 408, 412
client debugging, IKEv2, 450
client inside, 338
clients, remote access clients. *See* remote access clients
cloning, virtual access cloning, 295–297
cluster debugging, IKEv2, 450
cluster loads, FlexVPN load balancer, 369–372
CMD (Cisco meta data), 179
CoA (change-of-authorization), 303–304
co-existence, FlexVPN, authorization types, 245
collect command, 441
combined algorithm overhead, 512–513
combined mode ciphers, 77, 112
combined-mode ciphers, IKEv2 proposals, 110
commands
aaa authorization, 223, 225, 229, 233
aaa authorization group, 229, 237–238
aaa authorization group override, 294
aaa authorization user, 285, 484

aaa new-model command, 222
authenticate remote anyconnect-eap, 323
authentication, 147
backup group, 353
CFG_ACK, 252
CFG_REPLY, 251
CFG_REQUEST, 251
CFG_SET, 252
clear crypto ikev2 diagnose error, 461
clear crypto ikev2 sa, 186
clear crypto ikev2 sa remote, 360
clear crypto session, 186
client connect tunnel interface-number, 348
client connect Tunnelo, 408, 412
client inside, 338
collect, 441
config-exchange request, 251
config-exchange set accept, 251
config-exchange set send, 251
config-set, 497
copy run start, 346
crypto eap credential profile1, 337
crypto ikev2 authorization policy, 222, 228
crypto ikev2 cluster, 368
crypto ikev2 cookie-challenge number, 37
crypto ikev2 diagnose error, 159
crypto ikev2 disconnect-revoked-peers, 182
crypto ikev2 keyring, 129
crypto ikev2 name-mangler, 224
crypto ikev2 profile, 138
crypto ikev2 redirect client, 374
crypto ikev2 redirect gateway init, 377
crypto IPsec df-bit {clear | set | copy}, 528
crypto IPsec fragmentation, 532
crypto IPsec fragmentation {before-encryption | after-encryption}, 528
crypto IPsec profile, 348
crypto IPsec security-association replay window-size disable, 494
crypto key generate, 162
crypto mib IPsec flow history tunnel size, 427
crypto pki authenticate, 164

crypto pki profile enrollment, 163
debug aaa authorization, 436
debug aaa proto {local | radius}, 486–487
debug commands
AAA (authentication, authorization, and accounting), 501
EAP (Extensible Authentication Protocol), 501
IKEv2, 501
IPsec, 501
PKI (Public Key Infrastructure), 502
RADIUS, 501
debug crypto condition unmatched, 456
debug crypto ikev2, 187, 360, 436, 466, 473
debug crypto ikev2 client flexvpn, 353
debug crypto ikev2 cluster detail, 369
debug crypto ikev2 packet, 126, 450
debug crypto ikev2 packet debugging, 491
debug crypto ikev2 packet hexdump, 451
debug crypto IPsec, 453
debug crypto IPsec metadata sgt, 181
debug crypto kmi, 455
debug crypto pki, 473
debug ip dns name-list, 341
debug ip tcp, 474
debug vtemplate, 487
debug vtemplate cloning, 487
default crypto ikev2 authorization policy, 230
dn all, 227
dn common-name, 226
dn country, 227
dn domain, 226
dn locality, 227
dn organization, 227
dn organization-unit, 226
dn state, 227
eap all, 227
eap dn common-name, 227
eap dn country, 227
eap dn domain, 227
eap dn locality, 227
eap dn organization, 227

eap dn organization-unit, 227
eap prefix delimiter., 227
eap prefix delimiter @, 227
eap prefix delimiter (backslash), 227
eckeypair, 163
email all, 226
email domain, 226
email username, 226
enrollment url, 163
fqdn all, 226
fqdn domain, 226
fqdn hostname, 225
import all, 343
ip | ipv6 unnumbered, 102
ip address, 215
ip address negotiated, 334, 348
ip http server, 161
ip mtu, 499, 531
ip nat inside, 405
ip nat outside, 355, 405
ip tcp adjust-mss, 527
ip unnumbered, 218
ip unreachables, 520
ip vrf, 100
ip vrf forwarding, 100, 296
IPsec commands, considerations when moving to IKEv2, 543–544
ipv6 address, 215
ipv6 mtu, 531
ipv6 tcp adjust-mss, 527
ipv6 unnumbered, 218
ipv6 unreachables, 522
ivrf, 152
keyring aaa, 223, 225
match, 441
match certificate, 472
method-est, 163
monitor event-trace, 449
mtu, 518, 533
no crypto IPsec nat-transparency udp-encapsulation, 64
no lifetime, 184

no logging event link-status, 424
no route accept, 267
no shutdown, 161
peer reactivate, 346
pki trustpoint, 470
reconnect, 310
responder-only, 167
revocation-check method command, 164
route accept, 266, 498
route accept any, 230
route set interface, 230, 496
route set local, 267, 498
service-policy, 296
set ikev2-profile, 167
set mixed-mode, 99, 167
set peer hostname dynamic, 130
set pfs, 167
set reverse-route, 167
set security-association, 167
set security-association replay window-size disable, 494
set transform-set, 167
show aaa attribute protocol radius, 228
show cef interface, 531
show commands
  IKEv2, 500
  IPsec, 500
  PKI (Public Key Infrastructure), 501
  troubleshooting, 447
show crypto ikev2 authorization policy, 229
show crypto ikev2 client flexvpn, 348, 358
show crypto ikev2 client flexvpn flex1 detail, 342
show crypto ikev2 client flexvpn name, 358
show crypto ikev2 cluster, 369, 374
show crypto ikev2 diagnose error, 460
show crypto ikev2 proposal, 34, 109, 463
show crypto ikev2 sa detail, 282
show crypto ikev2 sa detailed, 144, 177, 313, 343, 359, 397, 399
show crypto ikev2 session detailed, 144
show crypto IPsec sa, 530, 560
show crypto IPsec transform-set, 488

show crypto pki certificate verbose, 480
show crypto pki certificates, 467
show crypto pki counters, 475
show crypto pki trustpoints, 467
show crypto session, 220, 559
show crypto session brief, 560
show crypto session detail, 186, 313
show crypto sessions, 95
show crypto sockets, 95
show derived-config, 91
show ip dhcp import, 343
show ip dns name-list, 341, 343
show ip interfaces, 518
show ip nat statistics, 406
show ip nat translations, 406
show ip route, 266–267
show ip route vrf, 460
show ip traffic, 522
show ipv6 interfaces, 518
show ipv6 traffic, 523
show ntp associations, 471
show platform hardware qfp active feature ipfrag global, 522–523
show run all, 184
show running-config, 91
show running-configuration, 478
show standby, 375
show track, 359
SNMP IKE trap commands, 427
SNMP IPsec trap commands, 438–439
snmp-server enable traps ike tunnel start, 425
snmp-server enable traps ike tunnel stop, 426
snmp-server enable traps IPsec tunnel start, 438
snmp-server enable traps IPsec tunnel stop, 439
snmp-server enable traps snmp linkdown linkup, 439–440
snmp-server enable traps snmp linkup linkdown, 424–425
test aaa, 481
tunnel destination, 215, 218
tunnel destination dynamic, 349, 412
tunnel destination peer-address, 90
tunnel mode, 215, 218

tunnel mode gre ip, 94
tunnel path-mtu-discovery, 499, 534–535
tunnel protection, 139, 167–168, 216, 218
tunnel protection IPsec, 123
tunnel source, 218
tunnel source dynamic, 348, 408
tunnel vrf name, 102
virtual-template 1 mode auto, 396
vrf definition, 100
vrf forwarding, 100, 216, 218, 486
vrf forwarding name, 102
components of
FlexVPN load balancer, 363
HSRP (hot standby routing protocol), 366–367
IKEv2 redirect, 363–366
IPsec, 5
PAD (Peer Authorization Database), 6
SAD (Security Association Database), 6
SPD (Security Policy Database), 5–6
SPI (Security Parameter Index), 5
Split-DNS, FlexVPN client, 340–343
conditional debugging, 456–457
confidentiality, 4
Config Payload, 58
config-exchange request, 251
config-exchange set accept, 251
config-exchange set send, 251
config-set command, 497
configuration attributes
auto-reconnect, FlexVPN server, 310–311
FlexVPN, 253–258
configuration constructs, IKEv2, 106
configuration examples
EAP (Extensible Authentication Protocol), FlexVPN server, 278–283
keyring, IKEv2, 134–135
configuration exchange, FlexVPN, 250
enabling, 250–251
examples, 259–264
configuration payload, 75
attributes, 59

FlexVPN, 251–253, 258
configuration payload exchange, 58–59
configurations, IKEv2, 106
configuring
AAA infrastructure, FlexVPN, 222–223
AAA-based pre-shared keys, FlexVPN server, 284
authorization, FlexVPN, 233
auto-reconnect, FlexVPN server, 313–315
backup-gateway attributes, 347
default-domain attributes, 344
Diffie-Hellman group, IKEv2, 113–114
dual-factor authentication, AnyConnect-EAP, 324–325
EAP (Extensible Authentication Protocol), FlexVPN server, 277–278
configuration examples, 278–283
FlexVPN, RADIUS servers, 388–390
FlexVPN client, dual-homed branch routers, 411–412
FlexVPN server, 387–388
IKEv2, encryption, 111–112
IKEv2 proposals, 108–111
under IKEv2 policies, 119–120
integrity, IKEv2, 113
IPsec configuration, 166–167
IPv4 DNS server attributes, 340
IPv4 WINS attribute, 343
IPv6 DNS server attributes, 341
keyring, IKEv2, 129–132
keys, in peer blocks, 132
match statements
under IKEv2 policies, 120–121
IKEv2 profiles, 139–142
name mangler, IKEv2, 224–227
peer blocks, in keyring, 130
peers, in peer blocks, 130–131
policies, IKEv2, 118–119
profiles, IKEv2, 138–139
pseudorandom function, IKEv2, 115
RADIUS change-of-authorization (CoA), 304
RADIUS Packet-of-Disconnect, FlexVPN server, 300
Split-DNS attributes, 341
static P2P tunnel interfaces, 214–216

trustpoints (TP), 476
user authentication, AnyConnect-EAP, 318–319
virtual-template interfaces, 216–219
connect auto mode, 408
connectionless integrity, 4
context-specific configuration, IKEv2, 106
continuous channel mode, 77
Cookie Challenge, IKEv2, 156
IKEv2 Cookie Challenge and Call Admission Control, 207–210
cookie notification, IKEv2, 36–38
copy run start, 346
Counter (CTR) mode, 510
CREATE_CHILD_SA, 24, 53, 53–54
CRL (certification revocation list), 181
CRL Distribution Point (CDP), 475
CRLs (Certificate Revocation Lists), 163
crypto eap credential profile1, 337
crypto ikev2 authorization policy, 222, 228
crypto ikev2 cluster, 368
crypto ikev2 cookie-challenge number, 37
crypto ikev2 diagnose error, 159
crypto ikev2 direct gateway init, 377
crypto ikev2 disconnect-revoked-peers, 182
crypto ikev2 error, 448
crypto ikev2 keyring, 129
crypto ikev2 name-mangler, 224
crypto ikev2 profile, 138
crypto ikev2 redirect client, 374
crypto IPsec df-bit {clear | set | copy}, 528
crypto IPsec error, 448
crypto IPsec fragmentation, 532
crypto IPsec fragmentation {before-encryption | after-encryption}, 528
crypto IPsec profile, 348
crypto IPsec security-association replay window-size disable, 494
crypto key generate, 162
crypto maps, 79
demise of, 86–87
versus tunnel protection, 80–81
crypto mib IPsec flow history tunnel size, 427

crypto pki authenticate, 164
crypto pki profile enrollment, 163
crypto pki server, 160
crypto sockets, 94–95
cryptographic exchange bloat, 77
cryptographic strength, 111–112
pre-shared keys, 135
cryptography
asymmetric cryptography, 11–12
IPsec VPNs, 7
asymmetric cryptography, 8
Diffie-Hellman exchange, 8–11
symmetric cryptography, 7–8
CTR (Counter) mode, 510

D

data encapsulation, GRE (generic routing encapsulation), 495
data encryption, 488
SNMP with IPsec, 437–439
data origin authentication, 4
data usage, monitoring, 440–443
datagram format, ESP (Encapsulating Security Payload), 18–19
dead peer detection, 59–61
Dead Peer Detection (DPD), IKEv2, 149–151, 158–159
debug aaa authorization, 436
debug aaa proto {local | radius}, 486–487
debug commands
AAA (authentication, authorization, and accounting), 501
EAP (Extensible Authentication Protocol), 501
IKEv2, 501
IPsec, 501
PKI (Public Key Infrastructure), 502
RADIUS, 501
debug crypto condition unmatched, 456
debug crypto ikev2, 123, 187, 360, 436, 466, 473
debug crypto ikev2 client flexvpn, 353, 450
debug crypto ikev2 cluster detail, 369, 375
debug crypto ikev2 packet, 126, 450
debug crypto ikev2 packet debugging, 491

debug crypto ikev2 packet hexdump, 451
debug crypto IPsec, 453
debug crypto IPsec metadata sgt, 181
debug crypto kmi, 123, 455
debug crypto pki, 473
debug ip dns name-list, 341
debug ip packet, 406
debug ip tcp, 474
debug vtemplate, 487
debug vtemplate cloning, 487
debugging, 449
authentication, with PKI, 470
conditional debugging, 456–457
FlexVPN client, 360
IKEv2, 449–453
IPsec, 488–491
IPsec debugging, 453
KMI (Key Management Interface), 453–455
PKI (Public Key Infrastructure), 456
verbose debugging, 181
default crypto ikev2 authorization policy, 230
default IKEv2 authorization policy, 229–231
default IKEv2 policies, 121–122
default proposals, IKEv2, 115–117
default-domain attributes, configuring, 344
defaults, smart defaults, 168–169
deleting, SAs (Security Associations), 57–58
deployments
IKEv1, 551–552
IKEv2
Cookie Challenge and Call Admission Control, 207–210
pre-shared-key authentication with smart defaults, 189–194
RSA authentication using HTTP URL lookup, 200–207
depo, IKEv2, ECDSA (Elliptic-Curve Digital Signature Algorithm) authentication, 194–200
DER (Distinguished Encoding Rules), 44, 201
DES, 111
detail keyword, 559
detection, NAT-D (Network Address Translation-Detection), 64
DF (Don’t Fragment), 172

diagnose error, IKEv2, 460–461
diagnostics, IKEv2, 159
dial backups, FlexVPN client, 352–353
Differentiated Services Code Point (DSCP), 74
Diffie, Whitfield, 9
Diffie-Hellman exchange, 8–11, 24, 26–29
initiators, 28
MITM (man-in-the-middle) attack, 45
IKE_SA_INIT, 29
Diffie-Hellman group, 30
IKEv2, configuring, 113–114
Diffie-Hellman tests, IKEv2, 65
digital certificates, 12
digital signatures, 12
IKEv2, 12–13
disabling
anti-replay, 494
default IKEv2 policies, 122
default IKEv2 proposals, 116
profiles, IKEv2, 153
disconnecting revoked peers, 182
DisconnectOnSuspend, 311
displaying, profiles, IKEv2, 153–154
Distinguished Encoding Rules (DER), 44, 201
distinguished name (DN), 224
DN (distinguished name), 224
extracting names from, 226–227
dn all, 227
dn common-name, 226
dn country, 227
dn domain, 226
dn locality, 227
dn organization, 227
dn organization-unit, 226
dn state, 227
domain names, FlexVPN client, 344–345
Don’t Fragment (DF), 172
DPD (Dead Peer Detection), IKEv2, 149–151, 158–159
DSCP (Differentiated Services Code Point), 74

dual stack
FlexVPN, 391–392
FlexVPN client, 335
GRE (generic routing encapsulation), 96
dual-factor authentication
AnyConnect-EAP, 320–324
AnyConnect-EAP XML messages, 322–324
configuring with AnyConnect-EAP, 324–325
dual-homed branch routers, FlexVPN client configuration, 408–409, 411–412
dummy packets, ESP (Encapsulating Security Payload) version 3, 20
dVTI (dynamic VTI), 92, 153
dynamic keyword, 346
dynamic routing, FlexVPN client, 335
dynamic routing protocols, 498–499
dynamic tunnel interfaces, 91–92
dynamic tunnel source, FlexVPN WAN resiliency, 407–408
dynamic VTI (dVTI), 92

E

EAP (Extensible Authentication Protocol), 48–50, 146, 480–485
authentication, 434–436
debug commands, 501
FlexVPN server, configuring, 277–278
examples, 278–283
eap all, 227
EAP authentication
FlexVPN client, 335, 337–338
FlexVPN server, 271–272
EAP methods, 272
steps for, 275–277
eap dn common-name, 227
eap dn country, 227
eap dn domain, 227
eap dn locality, 227
eap dn organization, 227
eap dn organization-unit, 227
eap dn state, 227
EAP identity, 224
extracting names, 227
FlexVPN server, 273–275

EAP message flow, FlexVPN server, 273
EAP methods
FlexVPN server, 272
TLS (transport layer security), 272
eap prefix delimiter., 227
eap prefix delimiter @, 227
eap prefix delimiter \, 227
EAP timeout, FlexVPN server, 275
EAP tunneled methods, 272
ECDH (Elliptic Curve Diffie Hellman), 10
ECDSA (Elliptic-Curve Digital Signature Algorithm), 73
ECDSA (Elliptic-Curve Digital Signature Algorithm) authentication
IKEv2, 194–200
troubleshooting, 465–468
ECDSA (Elliptic-Curve Digital Signature Algorithm) signatures, 12
eckeypair, 163
EEM (embedded event manager), 356–358
EIGRP (enhanced interior gateway routing protocol), 191
EKU (Extended Key Usage), 469, 480
AnyConnect, 469
Elliptic Curve Diffie Hellman (ECDH), 10
email
extracting names from, 226
IKE identity types, 224
email all, 226
email domain, 226
email username, 226
embedded event manager (EEM), 356–358
enabling, configuration exchange, FlexVPN, 250–251
encapsulating security payload overhead, 507–509
Encapsulating Security Payload (ESP). *See* ESP (Encapsulating Security Payload), 2–3
encapsulation, modes of encapsulation, 82
GRE encapsulation, 82–83
GRE over IPsec, 83
encipherment, 7
ENCR (Encryption algorithm), 30
encrypted payload structures, 43–44
encryption, 7, 14
ESP (Encapsulating Security Payload), 17–18

IKE_AUTH, 42–43
IKEv2
algorithms in Cisco IOS, 111
configuring, 111–112
IP Authentication Header, 15–16
Encryption Algorithm (ENCR), 30
encryption overhead, 510–511
enhanced interior gateway routing protocol (EIGRP), 191
Enrollment over Secure Transport (EST), 163
enrollment url, 163
error debugging, IKEv2, 450
ESN (Extended Sequence Numbers), 30
ESP (Encapsulating Security Payload), 2–3, 17, 504
anti-replay services, 18
authentication, 18
confidentiality, 4
datagram format, 18–19
encryption, 18
ESP (Encapsulating Security Payload) version 3, 19
dummy packets, 20
extended sequence numbers, 19
TFC (Traffic Flow Confidentiality), 20
ESP-NULL, 16
EST (Enrollment over Secure Transport), 163
event-trace monitoring, 447–449
examples
AAA-based pre-shared keys, FlexVPN server, 285–287
configuration exchange, FlexVPN, 259–264
EAP (Extensible Authentication Protocol), FlexVPN server, 278–283
FlexVPN, implicit authorization, 243–245
IKEv2 load balancing, FlexVPN load balancer, 376–378
IPsec configuration, 168
overhead, 516–517
RADIUS change-of-authorization (CoA), FlexVPN server, 305–309
RADIUS Packet-of-Disconnect, FlexVPN server, 301–303
soft migration, transitioning from IKEv1 to IKEv2, 551–552
virtual access cloning, 295–297
exchange modes
IKEv1, 70–71
IKEv2, 71–72

Exchange Type, 55
explicit padding, 510
Extended Authentication within IKE (XAUTH), 69
Extended Authentication within ISAKMP/Oakley, 69
Extended Key Usage (EKU), 469
extended sequence numbers, ESP (Encapsulating Security Payload) version 3, 19
Extensible Authentication Protocol. ***See*** **EAP (Extensible Authentication Protocol)**
extensions, for EAP-only authentication, 65
external AAA servers, FlexVPN, group authorization, 239–241
extra overhead, 516–517
extracting names
from DN identity, 226–227
from EAP identity, 227
from email identity, 226
from FQDN identity, 225–226
EzVPN, 333, 540
FlexVPN client, 336

F

familiarization, considerations when moving to IKEv2, 545
FIB (Forwarding Information BAse), 94
flapping tunnel interface, 499
Flexible NetFlow, 418
FlexVPN, 540
AAA-based pre-shared keys, 381–382
branch-1 router configuration, 382–383
branch-2 router configuration, 383
hub router configuration, 383–384
RADIUS server configuration, 384–386
attributes, 253
authorization, 231–233
configuring, 233
group authorization, 237–241
implicit authorization, 242–245
user authorization, 235–237
authorization types, 245–250
precedence, 247–250
auto-detection of tunnel encapsulation and transport, 219–221
building blocks of
Cisco IOS AAA infrastructure, 221–223

IKEv2, 213–214
point-to-point tunnel interfaces, 214–221
Cisco private use configuration attributes, 257–258
configuration attributes, 258
and authorization, 253–258
configuration exchange, 250
enabling, 250–251
examples, 259–264
configuration payload, 251–253, 258
configuring
AAA infrastructure, 222–223
RADIUS servers, 388–390
default IKEv2 authorization policy, 229–231
group and user authorization, 386
IKEv2, authorization policy, 228–229
migration strategies, 544
overview, 211–213
remote subnets, 264
learning from peer, 266–267
learning locally, 265–266
routing, 264–265
routing, dual stack, and tunnel mode auto, 391–392
spoke configuration
branch 1 configuration, 392–394
branch 2 configuration, 394–395
hub configuration, 395–397
verification at branch 1, 397–399
verification at branch 2, 399–400
verification on hub, 401–404
value proposition, 213
virtual access cloning examples, 295–297
FlexVPN backup tunnels, track-based tunnel activation, 414–415
FlexVPN client
advanced features, 336
backup gateway lists, 347
backup gateways, 346
backup groups, 353–354
building blocks of
FlexVPN client profiles, 334
IKEv2 configuration exchange, 334

NAT (Network Address Translation), 335
object tracking, 334
static P2P tunnel interfaces, 334
clearing IKEv2 FlexVPN client sessions, 360
configuring on dual-homed branch routers, 411–412
debugging, 360
dial backups, 352–353
domain names, 344–345
dual stack, 335
dual-homed branch routers, configuring, 408–409
dynamic routing, 335
EAP authentication, 335, 337–338
EzVPN, 336
FlexVPN load balancer, 374
group and user authorization
branch 1 configuration, 386
branch 2 configuration, 387
logs, 390–391
NAT (Network Address Translation), 354–355, 404–405
verification, 405–407
network extension modes, 336
overview, 332–333
PKI (Public Key Infrastructure), 356
pre-shared keys, 356
profiles, 345–346
reactivating peers, 346
resolution of FQDN (fully qualified domain names), 346
setting up FlexVPN servers, 336–337
Split-DNS, 338–340
components of, 340–343
tracking, 356
EEM (embedded event manager), 356–358
tracking lists of objects, with boolean expressions, 350–352
troubleshooting
debugging, 360
show commands, 358–359
tunnel destination, 349
tunnel initiation, 350
automatic mode, 350
manual mode, 350

track mode, 350
tunnel interface, 347–348
tunnel source, 348–349
verification, 409–410, 412–413
WINS (Windows Internet Naming Service), 343–344
FlexVPN client profiles, 334
FlexVPN feature, 90
FlexVPN hub resiliency, backup peers, 411
FlexVPN IKEv2 Load Balancer, 367–369
FlexVPN load balancer
cluster loads, 369–372
components of, 363
HSRP (hot standby routing protocol), 366–367
IKEv2 redirect, 363–366
FlexVPN client, 374
FlexVPN IKEv2 Load Balancer, 367–369
IKEv2 load balancing, examples, 376–378
IKEv2 redirect, 372–373
redirect loops, 373–374
troubleshooting, IKEv2 load balancing, 374–375
FlexVPN server
AAA-based pre-shared keys, 283–284
configuring, 284
examples, 285–287
RADIUS attributes, 285
accounting, 287–290
AnyConnect-EAP, 315–316
configuring dual-factor authentication, 324–325
configuring user authentication, 318–319
dual-factor authentication, 320–324
user authentication, 316–318
auto detection of tunnel transport and encapsulation, 297–298
configuring, 387–388
EAP (Extensible Authentication Protocol)
configuration examples, 278–283
configuring, 277–278
EAP authentication, 271–272
EAP methods, 272
steps for, 275–277
EAP identity, 273–275

EAP message flow, 273
EAP timeout, 275
IKEv2 auto-reconnect, 309–310
configuration attributes, 310–311
configuring, 313–315
smart DPD, 311–313
per-session interface, 290–291
query-identity, 277
RADIUS attributes, 325–329
RADIUS change-of-authorization (CoA), 303–304
configuring, 304
examples, 305–309
RADIUS Packet-of-Disconnect, 299–300
configuring, 300
examples, 301–303
remote access clients, 329
Cisco IKEv2 AnyConnect clients, 330
Microsoft Windows 7 IKEv2 clients, 329–330
sequence of events, 270–271
setting up, 336–337
timeout option, 278
virtual access configurations
AAA authorization, 293–294
incoming sessions, 294
virtual templates, 291–293
FlexVPN WAN resiliency
dual-homed branch routers, FlexVPN client configuration, 408–409
dynamic tunnel source, 407–408
FlexVPN client, verification, 409–410
flow monitors, 441
flow records, 441
Forwarding Information Base (FIB), 94
FQDN (fully qualified domain name), 224
extracting names from, 225–226
resolution of FQDN (fully qualified domain names), FlexVPN client, 346
fqdn all, 226
fqdn domain, 226
fqdn hostname, 225
fragmentation
IKEv2, 171–172, 173–178

session authentication, 181–182
session deletion on certificate expiry, 184
session deletion on certificate revocation, 182–184
session lifetime, 185–187
IP fragmentation, 172–173
IPsec
impact of, 535–536
IPv4, 519–522
IPv6, 522–523
MTU (maximum transmission unit), 518–519
PMTUD (path MTU discovery), 523–525, 527–531
TCP MSS clamping, 525
tunnels, 531
GRE (generic routing encapsulation), 532–533
GRE over IPsec, 534
IPsec only (VTI), 531–532
PMTUD (path MTU discovery), 534–535
Front-door VRF (VFRF), 118

G

generic routing encapsulation. *See* GRE (generic routing encapsulation)
GET VPN, 212–213
global configuration, IKEv2, 106, 155
HTTP URL-based certificate lookup, 156
global IKE, considerations when moving to IKEv2, 543–544
GRE (generic routing encapsulation), 80, 495
fragmentation, tunnels, 532–533
implementation modes
auto tunnel mode, 99
dual stack, 96
mixed mode, 96–99
mGRE (multipoint GRE), 92–94
overhead, 505–507
traffic, non-IP protocols, 86
traffic selectors, 51–52
tunnel configuration, 88–89
VRF aware, 101–102
GRE encapsulation, 82–83
GRE over IPsec, 83
fragmentation, tunnels, 534

IPsec transport mode, 83–84
IPsec tunnel mode, 84–85
VRF aware, 102–103
GRE tunnel keys, mismatching, 495
GRE/IP encapsulation, 82
GRE/IPsec, 87–88
group authorization
FlexVPN, 237–241
precedence, 249–250
FlexVPN client, 386
branch 1 configuration, 386
branch 2 configuration, 387
guiding principles, IKEv2, 106

H

half connections, 36
hard migration, transitioning from IKEv1 to IKEv2, 548–549
hardware limitations, considerations when moving to IKEv2, 540
Hardware Security Modules (HSM), 545
hash algorithms, signatures, 163
Hashed Message Authentication Code (HMAC), 4
HDR, 26, 76
Hellman, Martin, 9
high availability, 74
considerations when moving to IKEv2, 547
history of, IKEv1, 67–69
HMAC (Hashed Message Authentication Code), 4, 18, 40
integrity overhead, 511–512
HMAC-MD5–96, 515
HMAC-SHA-1–96, 513, 515
HMAC-SHA-256–128, 515
HMAC-SHA-384–192, 515
HMAC-SHA-512–256, 515
hot standby routing protocol (HSRP), FlexVPN load balancer, 366–367
HSM (Hardware Security Modules), 545
HSRP (hot standby routing protocol), FlexVPN load balancer, 366–367
HTTP URL lookup, RSA authentication using HTTP URL lookup, 200–207
HTTP URL-based certificate lookup, 156
HTTP_CERT_LOOKUP_SUPPORTED, 39

hub configuration, FlexVPN, spoke configuration, 395–397
hub router configuration, AAA-based pre-shared keys, FlexVPN, 383–384
hub-and-spoke topology
hard migration, 549
migration strategies, 562–564

I

ICV (Integrity Check Value), 4, 15
ID_DER_ASN1_DN, 45
ID_DER_ASN1_GN, 45
ID_IPV4_ADDR, 44
ID_IPV6_ADDR, 45
ID_KEY_ID, 45
ID_RFC822_ADDR, 45
identities, 44–45
EAP identity, FlexVPN server, 273–275
IKEv2, 74
local IKE identities, defining local IKE identities, 143–145
matching peers, 141–142
identity local, 143–144
identity services engine (ISE), 178
identity times, IKE identity types, 224
IETF (Internet Engineering Task Force), 23
IKE (Internet Key Exchange), 2
IKE identity types, 224
IKE trap commands, SNMP (Simple Network Management Protocol), 427
IKE_AUTH, 13, 24, 42, 174, 175–176
authentication, 42–43
encryption, 42–43
identity, 45
parameters, 43
traffic selectors, 50–52
troubleshooting, 464
ECDSA (Elliptic-Curve Digital Signature Algorithm) authentication, 465–468
RSA authentication, 465–468
IKE_SA_INIT, 25–26, 124
IKEv1
anti-denial of service, 72
authentication, 73–74
combined mode ciphers, 77

configuration payload, 75
continuous channel mode, 77
cryptographic ciphers, 77
deployments, 551–552
exchange modes, 70–71
high availability, 74
history of, 67–69
lifetime, 72–73
matching on identity, 75–76
NAT (Network Address Translation), 74–75
traffic selectors, 74
IKEv2 (Internet Key Exchange protocol version 2), 1
anti-denial of service, 72
authentication, 14, 45, 73–74
EAP (Extensible Authentication Protocol), 48–50
pre-shared-key-based authentication, 47
signature-based authentication, 46
authorization policy, 228–229
auto-reconnect
FlexVPN server, 309–310
FlexVPN server, configuration attributes, 310–311
FlexVPN server, configuring, 313–315
FlexVPN server, smart DPD, 311–313
CAC (Call Admission Control), 157
certificate requests, 38–39
HTTP_CERT_LOOKUP_SUPPORTED, 39
clearing IKEv2 FlexVPN client sessions, 360
client debugging, 450
cluster debugging, 450
combined mode ciphers, 77
configuration constructs, 106
configuration exchange, FlexVPN client, 334
configuration payload, 75
considerations when moving to IKEv2
asymmetric routing, 547–548
authentication, 546–547
client awareness, 545
current capacity, 542–543
current VPN technology, 540–541
familiarization, 545

FlexVPN, 544
global IKE, 543–544
hardware limitations, 540
high availability, 547
IP addresses, 543
IPsec commands, 543–544
IPv6, 546
PKI (Public Key Infrastructure), 545–546
restrictions when running IKEv1 and IKEv2 simultaneously, 541–542
routing protocols, 541
software, 543
VPN gateways, 543
context-specific configuration, 106
continuous channel mode, 77
Cookie Challenge, 156
cryptographic exchange bloat, 77
debug commands, 501
debugging, 449–453
deployments
Cookie Challenge and Call Admission Control, 207–210
ECDSA (Elliptic-Curve Digital Signature Algorithm) authentication, 194–200
pre-shared-key authentication with smart defaults, 189–194
RSA authentication using HTTP URL lookup, 200–207
diagnose error, 460–461
diagnostics, 159
Diffie-Hellman group, configuring, 113–114
digital signatures, 12–13
DPD (Dead Peer Detection), 149–151, 158–159
dual-factor authentication, 320–321
encrypted payload structures, 43–44
encryption, 14
algorithms in Cisco IOS, 111
configuring, 111–112
error debugging, 450
exchange modes, 71–72
FlexVPN, 213–214
fragmentation, 171–172, 173–178
global configuration, 106, 155
HTTP URL-based certificate lookup, 156
guiding principles, 106

high availability, 74
identities, 44–45, 74
initial contact, 52, 151
integrity, configuring, 113
internal debugging, 450
key material generation, 39–42
keyring, 106, 128–129
configuration examples, 134–135
configuring, 129–132
key lookup on initiators, 132–133
key lookup on responders, 133–134
overview, 136
pre-shared-key authentication with smart defaults, 190
lifetime, 72–73
load balancing, 374–375
examples, 376–378
matching on identity, 75–76
MOBIKE, 75
name mangler, 223–224
configuring, 224–227
NAT (Network Address Translation), 61–64, 74–75
NAT keepalives, 152, 159
overview, 23–24
packet debugging, 450
packet structure, 55–56
PKI (Public Key Infrastructure)
examples, 164–166
trustpoints (TP), 163–164
policies, 106, 117–118
configuring, 118–119
configuring match statements, 120–121
configuring proposals under, 119–120
default IKEv2 policies, 121–122
policy configuration examples
multiple proposals, 126–127
per-peer IKEv2 policies, 125–126
policy selection
initiators, 122–124
responders, 124–125
pre-shared key lookup parameters, 134

profiles, 106, 136–137
configuring, 138–139
configuring match statements, 139–142
defining local and remote authentication methods, 145–149
defining local IKE identities, 143–145
defining scope, 143
disabling, 153
displaying, 153–154
initial contact, 151
initiators and responders, 154
IVRF (inside VRF), 152
lifetime, 151
matching peer identity, 142–143
matching peers by identity, 141–142
NAT keepalives, 152
overview, 154–155
peer authorization database, 137–138
pre-shared-key authentication with smart defaults, 190
virtual template interface, 153
proposals, 106, 107–108
configuring, 108–111
configuring under IKEv2 policies, 119–120
default proposals, 115–117
protected tunnel interface, 558
pseudorandom function, configuring, 115
public-private key pair, 162
redirect, FlexVPN load balancer, 363–366, 372–373
reliability, 77
request-response, 61
routing, 496–498
SAs (Security Associations)
lifetime, 151
rekey, 54–55
session authentication, 181–182
session deletion on certificate expiry, 184
session deletion on certificate revocation, 182–184
session lifetime, 185–187
SGT (security group tags), 178–181
shared secrets, 13
show commands, 500

smart defaults, 168–169
SNMP (Simple Network Management Protocol), 425–427
standard attributes, 253
syslog messages, 428–429
traffic selectors, 74
window size, 158
IKEv2 exchange, 24–25
cookie notification, 36–38
Diffie-Hellman exchange, 26–29
IKE_SA_INIT, 25–26
nonce, 35–36
SPI (Security Parameter Index), 34–35
IKEv2 SA, 555–556
IKEv2-password-local, 285
IKEv2-password-remote, 285
impact of, fragmentation, 535–536
implementation modes, GRE (generic routing encapsulation)
auto tunnel mode, 99
dual stack, 96
mixed mode, 96–99
implicit authorization, 269, 285
FlexVPN, 242–245
import all, 343
inacl, 304
incoming sessions, deriving virtual access configurations, FlexVPN server, 294
INFORMATIONAL exchange, 56
deleting SAs (Security Associations), 57–58
initial contact, IKEv2, 52, 151
initial exchange, 24
initial handshake, 24
INITIAL_CONTACT, 52
initialization vector (IV), 18
initiators
Diffie-Hellman exchange, 28
key lookup, 132–133
policy selection, IKEv2, 122–124
profiles, IKEv2, 154
inside VRF (IVRF), 118
INTEG (Integrity algorithm), 30
integrity, 15

connectionless integrity, 4
IKEv2, configuring, 113
Integrity Algorithm, 30
integrity algorithms, 511–512
Integrity Check Value (ICV), 4, 15
integrity overhead, 511–512
interface Tunnel number, 91
interface-config AAA attribute, 293–294
interface-config attribute, 296, 303
interfaces
dynamic tunnel interfaces, 91–92
flapping tunnel interface, 499
P2P (point-to-point) tunnel interfaces, 214–221
per-peer P2P tunnel interfaces, 221
static P2P tunnel interfaces, 214–216
static tunnel interfaces, 90
traffic selection by routing, 88–90
virtual access interfaces, 217
virtual interfaces, 87–88
virtual template interface, 291–293
configuring, 216–219
virtual-access interface, 290–291
internal debugging, IKEv2, 450
INTERNAL_IP4_ADDRESS, 254
INTERNAL_IP4_DNS, 256
INTERNAL_IP4_NBNS, 257
INTERNAL_IP4_NETMASK, 254
INTERNAL_IP4_SUBNET, 255
INTERNAL_IP6_ADDRESS, 255
INTERNAL_IP6_DNS, 256
INTERNAL_IP6_SUBNET, 256
Internet Engineering Task Force. *See* IETF (Internet Engineering Task Force)
Internet Key Exchange (IKE), 2
Internet Key Exchange protocol version 2. *See* IKEv2
ip | ipv6 unnumbered, 102
ip address, 215
ip address negotiated, 334, 348
IP addresses, considerations when moving to IKEv2, 543
IP Authentication Header, 15–16

IP connectivity, 423–424
troubleshooting, 457–460
IP fragmentation, overview, 172–173
ip http server, 161
ip mtu command, 499, 531
ip nat inside, 405
ip nat outside command, 355, 405
IP protocol numbers, 50
ip tcp adjust-mss, 527
ip unnumbered, 218
ip unreachables, 520
ip vrf, 100
ip vrf forwarding, 100, 296
IP_FQDN, 44
IPsec
anti-replay services, 491–494
building blocks of, 2
access control, 4
anti-replay services, 4
confidentiality, 4
connectionless integrity, 4
data origin authentication, 4
key management protocol, 3
SAs (Security Associations), 3
security protocols, 2–3
security services, 3
TFC (Traffic Flow Confidentiality), 4–5
components of, 5
PAD (Peer Authorization Database), 6
SAD (Security Association Database), 6
SPD (Security Policy Database), 5–6
SPI (Security Parameter Index), 5
debug commands, 501
debugging, 488–491
fragmentation
impact of, 535–536
IPv4, 519–522
IPv6, 522–523
MTU (maximum transmission unit), 518–519
PMTUD (path MTU discovery), 523–525, 527–531

TCP MSS clamping, 525
GRE over IPsec, 83
modes of, 20
transport mode, 20–21
tunnel mode, 21
overlay routing, 495
show commands, 500
VRF (Virtual Routing and Forwarding), 99–101
VRF aware, 101–102
IPsec commands, considerations when moving to IKEv2, 543–544
IPsec configuration, 166–167
examples, 168
smart defaults, 168–169
IPsec debugging, 453
IPsec dVTI, 153
IPsec mode overhead (without GRE), 505
IPsec overhead, 504–505
encapsulating security payload overhead, 507–509
examples, 516–517
GRE overhead, 505–507
IPsec mode overhead (without GRE), 505
plaintext MTU, 513–514
IPsec profiles, 167
IPsec Remote Access Client (IRAC), 58
IPsec Remote Access Server (IRAS), 58
IPsec SA
creating, 53–54
rekey, 54
traffic selectors, 51
IPsec security services, 3
IPsec SNMP trap, 437
IPsec transport mode, GRE over IPsec, 83–84
IPsec tunnel encapsulation, 92
IPsec tunnel mode
GRE over IPsec, 84–85
GRE/IPsec, 87–88
VTI (Virtual Tunnel Interface), 87–88
IPsec VPN methodology, troubleshooting, 446
IPsec VPNs, 2
cryptography, 7

asymmetric cryptography, 8
Diffie-Hellman exchange, 8–11
symmetric cryptography, 7–8
IPsec-v3 standards, 504
IPv4
IPsec, fragmentation, 519–522
tunnels, mixed mode, 96
IPv4 DNS server attributes, configuring, 340
IPv4 WINS attribute, 343
ipv4-pool, 250
IPv6
considerations when moving to IKEv2, 546
IPsec, fragmentation, 522–523
pre-shared-key authentication with smart defaults, 191
ipv6 address, 215
IPv6 DNS server attributes, configuring, 341
ipv6 mtu, 531
ipv6 tcp adjust-mss, 527
ipv6 unnumbered, 218
ipv6 unreachables, 522
IRAC (IPsec Remote Access Client), 58
IRAS (IPsec Remote Access Server), 58
irvf command, 152
ISAKMP, 68
ISE (identity services engine), 178
IV (initialization vector), 18
IVRF (inside VRF), 118
profiles, IKEv2, 152

K

KE (Key Exchange) payload, 27–28
keepalives, 24, 59–61
NAT keepalives, 152, 159
KEi, 26
KEr, 26
Key Exchange payload. ***See*** **KE (Key Exchange) payload**
key lookup
initiators, 132–133
responders, 133–134

Key Management Interface. ***See*** **KMI (Key Management Interface)**
key management protocol, 3
key material generation, IKEv2, 39–42
key pairs, public-private key pair, 162
Key Usage, 469
KEY_ENG_DELETE_SAS, 454
KEY_ENG_IPSEC_READY, 454
KEY_ENG_NOTIFY_INCR_COUNT, 454
KEY_ENG_REQUEST_SAS, 454
KEY_MGR_CREATE_IPSEC_SAS, 454
KEY_MGR_DELETE_SAS, 455
KEY_MGR_IKMP_READY, 454
KEY_MGR_SESSION_CLOSED, 455
KEY_MGR_VALIDATE_IPSEC_PROPOSALS, 454
keyring, IKEv2, 106, 128–129
configuration examples, 134–135
configuring, 129–132
configuring peer blocks, 130
key lookup on initiators, 132–133
key lookup on responders, 133–134
overview, 136
pre-shared-key authentication with smart defaults, 190
keyring aaa, 223, 225
keys
AAA-based pre-shared keys
FlexVPN, 381–382
FlexVPN server, 284
RADIUS attributes, 285
asymmetric keys, 132
authentication pre-shared keys, 429–431
configuring, in peer blocks, 132
GRE tunnel keys, mismatching, 495
pre-shared keys, 478–479
cryptographic strength, 135
FlexVPN client, 356
symmetric keys, 132
KeyUsage extension, 469
keywords
cached, 234
detail, 559

dynamic keyword, 346
max-redirects, 374
sign, 476
timeout, 484
KMI (Key Management Interface), debugging, 453–455

L

legacy algorithms, 111
lifetime, 72–73
IKEv2, SAs (Security Associations), 151
SAs (Security Associations), 7
session lifetime, 185–187
lifetime certificate, 184
liveness checking, 24
load balancers, FlexVPN load balancer. ***See*** **FlexVPN load balancer**
load balancing
high availability, 547
troubleshooting, IKEv2 load balancing, 374–375
local AAA database, FlexVPN, group authorization, 238–239
local authentication methods, IKEv2 profiles, 145–149
local IKE identities, defining, 143–145
logs, FlexVPN client, 390–391

M

Main mode, 70
Management Information Base (MIB), 419
man-in-the-middle (MITM) attack, 13
manual mode, tunnel initiation (FlexVPN client), 350
maps, crypto maps, 86–87
master session key (MSK), 272
match address local, 142–143
match certificate command, 472
match certificates, 139, 140–141
match command, 441
match fvrf, 142–143
match fvrf any statement, 122
match identity, 139, 141–142
match identity remote any, 142
match statements, configuring

under IKEv2 policies, 120–121
in IKEv2 profiles, 139–142
matching
peer identity, 142–143
peers
with certificate maps, 472–473
by certificates, 140–141
by identity, 141–142
matching on identity, 75–76
maximum authentication header overhead, 516
maximum ESP overhead, 515
maximum ICV padding, 514
maximum input padding, 514
maximum output overhead, 514
maximum overhead, 514–515
Maximum Segment Size (MSS), 525–527
maximum transmission unit (MTU), 172
IPsec, fragmentation, 518–519
max-redirects, 374
MD5, 115
Md5, 113
messages, Proposal Incomplete message, 108
method-est, 163
methodologies, monitoring methodology, 422–423
mGRE, 83
mGRE (multipoint GRE), 92–94
MIB (Management Information Base), 419
Microsoft Windows 7 IKEv2 clients, FlexVPN server, 329–330
migration strategies, 539, 548
considerations when moving to IKEv2
asymmetric routing, 547–548
authentication, 546–547
client awareness, 545
current capacity, 542–543
current VPN technology, 540–541
familiarization, 545
FlexVPN, 544
global IKE, 543–544
hardware limitations, 540

high availability, 547
IP addresses, 543
IPsec commands, 543–544
IPv6, 546
PKI (Public Key Infrastructure), 545–546
restrictions when running IKEv1 and IKEv2 simultaneously, 541–542
routing protocols, 541
software, 543
VPN gateways, 543
remote access, 565–566
topologies, 561
hub-and-spoke topology, 562–564
site-to-site, 561–562
transitioning from IKEv1 to IKEv2, 548
hard migration, 548–549
soft migration, 549–559
verification, 559–561
migration verification, 559–561
mismatching, GRE tunnel keys, 495
MITM (man-in-the-middle) attack, 13
Diffie-Hellman exchange, 45
mixed mode, GRE (generic routing encapsulation), 96–99
MMx, 70
MOBIKE, 75
Mobility and Multihoming protocol, 75
modes, 79
continuous channel mode, 77
modes of encapsulation, 82
GRE encapsulation, 82–83
GRE over IPsec, 83
modes of IPsec, 20
transport mode, 20–21
tunnel mode, 21
modifying
default IKEv2 policies, 122
default IKEv2 proposals, 116–117
monitor even-trace crypto IPsec, 448
monitor event-trace, 449
monitor event-trace crypto ikev2, 448
monitoring

AAA (authentication, authorization, and accounting), 418
authentication EAP, 434–436
authentication PKI, 431–434
authentication pre-shared keys, 429–431
authorization, RADIUS-based AAA, 436
data encryption, SNMP with IPsec, 437–439
data usage, 440–443
IP connectivity, 423–424
NetFlow, 418–419
overlay routing, 439–440
SNMP (Simple Network Management Protocol), 419–420
syslog, 421
VPN tunnel establishment, 425
Cisco IPsec flow monitor MIB, 425
SNMP with IKEv2, 425–427
monitoring methodology, 422–423
MPLS (Multiprotocol Label Switching), 2
MSK (master session key), 272
MSS (Maximum Segment Size), 525–526
adjustments, 526–527
MSS clamping, 525
MTU (maximum transmission unit), 172
IPsec, fragmentation, 518–519
plaintext MTU, 513–514
mtu command, 518, 533
multicast traffic, 85–86
multiple proposals, IKEv2 policy configuration examples, 126–127
multipoint GRE (mGRE), 92–94
Multiprotocol Label Switching. *See* MPLS (Multiprotocol Label Switching
multi-SA dVTI, 92

N

name mangler, IKEv2, 223–224
configuring, 224–227
name verification, AnyConnect, 468
names, extracting
from DN identity, 226–227
from EAP identity, 227
from email identity, 226
from FQDN identity, 225–226

NAPT (network address port translation), 221
NAT (Network Address Translation), 74–75
FlexVPN client, 335, 354–355, 404–405
verification, 405–407
IKEv2, 61–64
keepalives, 59–61
NAT keepalives, 59–61, 152, 159
NAT-D (Network Address Translation-Detection), 64
negotiations, SGT (security group tags), IKEv2, 178–181
NetFlow, 418–419, 440–441
network address port translation (NAPT), 221
network extension modes, FlexVPN client, 336
Network Time Protocol (NTP), 471
Next Hop Resolution Protocol (NHRP), 86, 93
next-generation encryption (NGE), 112
NGE (next-generation encryption), IKEv2, 125
NHRP (Next Hop Resolution Protocol), 86, 93
Ni, 26
no crypto IPsec nat-transparency udp-encapsulation, 64
no lifetime, 184
no logging event link-status, 424
no route accept, 267
no shutdown command, 161
non-broadcast, 93
nonce, IKEv2 exchange, 35–36
non-IP protocols, 86
Notification payload, 56
notifications, REDIRECT notification, 363–366
Nr, 26
NTP (Network Time Protocol), 471
null encryption, 16

O

OAKLEY, 68
object tracking, FlexVPN client, 334
EEM (embedded event manager), 356–358
OCSP (online certificate status protocol), 181
on-demand mode, DPD (Dead Peer Detection), 150
online certificate status protocol (OCSP), 181

outacl, 304
outbound IPsec SA parameters, 215
overhead
authentication header overhead, 509–510
combined algorithm overhead, 512–513
encryption overhead, 510–511
extra overhead, 516–517
GRE (generic routing encapsulation), 505–507
integrity overhead, 511–512
IPsec. ***See*** IPsec overhead
maximum authentication header overhead, 516
maximum ESP overhead, 515
maximum overhead, 514–515
plaintext MTU, 513–514
overlay routing, 439–440, 495
overload limit, 372

P

P2P (point-to-point) tunnel interfaces, 214–221
packet debugging, IKEv2, 450
packet structure, IKEv2, 55–56
Packet-of-Disconnect (PoD), 299
PAD (Peer Authorization Database), 6
IKEv2, profiles, 137–138
parameters
global configuration, IKEv2, 155
IKE_AUTH, 43
outbound IPsec SA parameters, 215
pre-shared key lookup parameters, IKEv2, 134
profiles, IKEv2, 136–137
path MTU discovery (PMTUD), 172–173
IPsec, fragmentation, 523–525
Peer Authorization Database. *See* PAD (Peer Authorization Database)
peer blocks, configuring
in keyring, 130
keys, 132
peers, 130–131
peer identity, matching, IKEv2, 142–143
peer reactivate, 346
peers, 2

backup peers, FlexVPN hub resiliency, 411
configuring, in peer blocks, 130–131
matching
with certificate maps, 472–473
by certificates, 140–141
by identity, 141–142
reactivating, FlexVPN client, 346
remote subnets, FlexVPN, 266–267
revoked peers, disconnecting, 182
PEM (Privacy Enhanced Mail), 201
periodic mode, DPD (Dead Peer Detection), 150
per-peer IKEv2 policies, 125–126
per-peer P2P tunnel interfaces, 221
per-session interface, FlexVPN server, 290–291
PKI (Public Key Infrastructure), 11, 159–160, 456
authentication, 431–434
CA (certificate authority), 12, 160–162
certificate-based authentication, 147–148
considerations when moving to IKEv2, 545–546
debug commands, 502
debugging, 456
authentication, 470
digital certificates, 12
examples, 164–166
FlexVPN client, 356
public-key cryptography, 11–12
public-private key pair, 162
show commands, 501
trustpoints (TP), 163–164
PKI trustpoints, 148–149, 470
plaintext MTU, 513–514
PMTUD (path MTU discovery), 172–173
fragmentation, tunnels, 534–535
IPsec, fragmentation, 523–525, 527–531
PoD (Packet-of-Disconnect), FlexVPN server, 299
point-to-point tunnel interfaces. ***See*** **P2P (point-to-point) tunnel interfaces**
policies, IKEv2, 106, 117–118
authorization policy, 228–229
configuring, 118–119
configuring match statements, 120–121

configuring proposals under, 119–120
default IKEv2 policies, 121–122
policy configuration examples, IKEv2
multiple proposals, 126–127
per-peer IKEv2 policies, 125–126
policy selection, IKEv2
initiators, 122–124
responders, 124–125
precedence, FlexVPN
group authorization, 249–250
user authorization, 247–249
pre-shared key authentication, 147
pre-shared key lookup parameters, IKEv2, 134
pre-shared keys, 13, 478–479
cryptographic strength, 135
FlexVPN client, 356
pre-shared-key authentication with smart defaults, 189–194
pre-shared-key-based authentication, 47
PRF (pseudorandom function), 7, 30, 31, 40–41
algorithms, 41
IKEv2, configuring, 115
Privacy Enhanced Mail (PEM), 201
profiles
FlexVPN client, 345–346
IKEv2, 106, 136–137
configuring, 138–139
configuring match statements, 139–142
defining local and remote authentication methods, 145–149
defining local IKE identities, 143–145
defining scope, 143
disabling, 153
displaying, 153–154
initial contact, 151
initiators and responders, 154
IVRF (inside VRF), 152
lifetime, 151
matching peer identity, 142–143
matching peers by identity, 141–142
NAT keepalives, 152
overview, 154–155

peer authorization database, 137–138
pre-shared-key authentication with smart defaults, 190
virtual template interface, 153
IPsec, 167
Proposal Incomplete message, 108
proposals
IKEv2, 106, 107–108
configuring, 108–111
configuring under IKEv2 policies, 119–120
default proposals, 115–117
multiple proposals, IKEv2 policy configuration examples, 126–127
Security Association Proposals, 29–34
protected tunnel interface, IKEv2, 558
proto id, 491
protocols
AAA (authentication, authorization, and accounting), 418
AH (Authentication Header), 2–3
Authentication Header (AH), 15
CDP (Cisco Discovery Protocol), 86
dynamic routing protocols, 498–499
EAP (Extensible Authentication Protocol). ***See*** EAP (Extensible Authentication Protocol), 48–50
ESP (Encapsulating Security Payload), 2–3, 17
HSRP (hot standby routing protocol), FlexVPN load balancer, 366–367
ISAKMP, 68
NetFlow, 418–419
NHRP (Next Hop Resolution Protocol), 86, 93
non-IP protocols, 86
OAKLEY, 68
OCSP (online certificate status protocol), 181
RFC4301, 3
routing protocols, considerations when moving to IKEv2, 541
SKEME, 68
SNMP (Simple Network Management Protocol), 419–420
SXP (Security group tag exchange), 179
syslog, 421
UDP (User Datagram Protocol), 25
pseudorandom function. *See* PRF (pseudorandom function)
PSK, authentication, 429
Public Key Infrastructure. *See* PKI (Public Key Infrastructure)
public-key cryptography, 8

PKI (Public Key Infrastructure), 11–12
public-private key pair, 162

Q

QCR (quantum computer resistant), 112
query, 337
query-identity, 274, 482
FlexVPN server, 277
Quick Mode, 70

R

RADIUS
debug commands, 501
PSK configuration, 478
RADIUS accounting, 287
authentication pre-shared keys, 429
RADIUS attributes
AAA-based pre-shared keys, 285
CoA (change-of-authorization), 303–304
FlexVPN server, 325–329
RADIUS change-of-authorization (CoA)
configuring, 304
FlexVPN server, 303–304
examples, 305–309
updating
session ACL, 307–309
session QoS policies, 305–307
RADIUS Packet-of-Disconnect, FlexVPN server, 299–300
configuring, 300
examples, 301–303
RADIUS servers
configuring
AAA-based pre-shared keys, 384–386
FlexVPN, 388–390
EAP (Extensible Authentication Protocol), configuration examples, 278, 280–281
reactivating peers, FlexVPN client, 346
reconnect, 310
Reconnect capable active session count, 315
Reconnect capable inactive session count, 315
ReconnectAfterResume, 311

reconnect-cleanup-interval, 311
reconnect-dpd-interval, 311
reconnect-session-id, 310
reconnect-timeout, 313
reconnect-token-id, 310
redirect, IKEv2, FlexVPN load balancer, 363–366, 372–373
redirect loops, FlexVPN load balancer, 373–374
redirect mechanisms, 65
REDIRECT notification, 363–366
REDIRECT payload, 65
re-enabling
default IKEv2 policies, 122
default IKEv2 proposals, 116
rekey, 7
IPsec SA, 54
SAs (Security Associations), IKEv2, 54–55
reliability, 77
remote access, migration strategies, 565–566
remote access clients, FlexVPN server, 329
Cisco IKEv2 AnyConnect clients, 330
Microsoft Windows 7 IKEv2 clients, 329–330
remote authentication methods, IKEv2 profiles, 145–149
remote subnets, FlexVPN, 264
learning from peer, 266–267
learning locally, 265–266
request-response, IKEv2, 61
resolution of FQDN (fully qualified domain names), FlexVPN client, 346
responder-only, 167
responders
key lookup, 133–134
policy selection, IKEv2, 124–125
profiles, IKEv2, 154
restoring modified default IKEv2 proposals, 117
restrictions when running IKEv1 and IKEv2 simultaneously, 541–542
revocation, certificate revocation, 473–476
revocation-check method command, 164
revoked peers, disconnecting, 182
RFC (Request for Comments), 23
RFC791, 520

RFC2401, 68
RFC2402, 68
RFC2403, 68
RFC2404, 68
RFC2405, 68
RFC2406, 68
RFC2407, 68
RFC2408, 68
RFC2409, 68, 73
RFC2410, 68
RFC2411, 68
RFC2412, 68
RFC2459, 469
RFC3164, 421
RFC3526, 11
RFC3706, 69
RFC3715, 69
RFC3748, 48, 73
RFC3947, 69
RFC3948, 69
RFC4301, 3, 137
RFC4302, 15
RFC4304, 69
RFC4478, 182
RFC4555, 75
RFC4739, 321
RFC4754, 69, 73
RFC4821, 525
RFC4945, 469
RFC5114, 11
RFC5685, 65
RFC5716, 299
RFC5998, 65
RFC6023, 66
RFC6989, 65
RFC7296, 44, 65
RFC7383, 174
RIB (Routing Information Base), 440, 495
Rivest-Shamir-Adleman (RSA) key pair, 160

Rivest-Shamir-Adleman Signature, 12
route accept, 266, 498
route accept any, 230
route set interface, 230, 496
route set interface statement, 497
route set local, 267, 498
routing
asymmetric routing, considerations when moving to IKEv2, 547–548
FlexVPN, 264–265, 391–392
IKEv2, 496–498
overlay routing, 439–440, 495
static routing, 496
traffic selection, 88–90
routing adjacency, 498
Routing Information Base (RIB), 440, 495
routing protocols, considerations when moving to IKEv2, 541
RSA (Rivest-Shamir-Adleman) key pair, 160
RSA authentication, troubleshooting, 465–468
RSA authentication using HTTP URL lookup, 200–207

S

IKE_SA_INIT, 24, 174–175
Diffie-Hellman exchange, 29
parameters, 26
IKE_SA_INIT exchange, troubleshooting, 461–464
SAD (Security Association Database), 6
SAil, 26
SArl, 26
SAs (Security Associations), 2–3
creating, 53–54
deleting, 57–58
IKEv2
lifetime, 151
rekey, 54–55
lifetime, 7
rekey, 54
scope, IKEv2, profiles, 143
Security Association Database. ***See*** **SAD (Security Association Database)**
Security Association Proposals, 29–34

Security Association (SA), creating, 53–54
Security Associations. *See* SAs (Security Associations)
Security Associations (SA), deleting, 57–58
Security group tag exchange (SXP), 179
security group tags. *See* SGT (security group tags)
Security Information Event Management. *See* SIEM (Security Information Event Management)
security levels
IKEv2 syslog messages, 428–429
syslog, 421
Security Parameter Index (SPI), 3, 5
security payload overhead, encapsulating, 507–509
Security Policy Database. *See* SPD (Security Policy Database)
security protocols, 2–3
security services, IPsec, 3
selecting, trustpoints (TP), 476–477
sequence of events, FlexVPN server, 270–271
service-policy, 296
service-policy input, 304
session ACL, updating, with RADIUS CoA, 307–309
session authentication, IKEv2, 181–182
session deletion on certificate expiry, 184
session deletion on certificate revocation, IKEv2, 182–184
session lifetime, IKEv2, 185–187
session QoS policies, updating, with RADIUS CoA, 305–307
set ikev2-profile, 167
set mixed-mode, 99, 167
set peer hostname dynamic, 130
set pfs, 167
set reverse-route, 167
set security-association, 167
set security-association replay window-size disable, 494
set transform-set, 167
SET_WINDOW_SIZE notification payload, 158
SGT (security group tags), 171
IKEv2, 178–181
SHA1, 115
Sha1, 113
SHA256, 115
SHA384, 115

Sha384, 113
SHA521, 115
Sha521, 113
shared secrets, 13
shared-key-based authentication, 47
show aaa attribute protocol radius, 228
show cef interface, 531
show commands
IKEv2, 500
IPsec, 500
PKI (Public Key Infrastructure), 501
troubleshooting, 447
FlexVPN client, 358–359
show crypto ikev2 authorization policy, 229
show crypto ikev2 client flexvpn, 358
show crypto ikev2 client flexvpn flex1 detail, 342
show crypto ikev2 client flexvpn name, 358
show crypto ikev2 cluster, 369, 374
show crypto ikev2 diagnose error, 460
show crypto ikev2 flexvpn, 348
show crypto ikev2 proposal, 34, 109, 463
show crypto ikev2 sa detail, 282
show crypto ikev2 sa detailed, 144, 177, 313, 343, 359, 397, 399
show crypto ikev2 session detailed, 144
show crypto ikev2 stats reconnect, 315
show crypto IPsec sa, 530, 560
show crypto IPsec transform-set, 488
show crypto pki certificate verbose, 480
show crypto pki certificates, 467
show crypto pki counters, 475
show crypto pki trustpoints, 467
show crypto session, 220, 559
show crypto session brief, 560
show crypto session detail, 186, 313
show crypto sessions, 95
show crypto sockets, 95
show derived-config, 91
show ip dhcp import, 343
show ip dns name-list, 341, 343

show ip interfaces, 518
show ip nat statistics, 406
show ip nat translations, 406
show ip route, 266, 267
show ip route vrf, 460
show ip traffic, 522
show ipv6 interfaces, 518
show ipv6 traffic, 523
show ntp associations, 471
show platform hardware qfp active feature ipfrag global, 522–523
show run all, 184
show running-config, 91
show running-configuration command, 478
show standby, 375
show track, 359
SIA (subject information access), 202
SIEM (Security Information Event Management), 417
sign keyword, 476
signature-based authentication, 46
signatures
digital signatures, IKEv2, 12–13
hash algorithms, 163
Simple Network Management Protocol (SNMP), 419–420
site-to-site, migration strategies, 561–562
SKEME, 68
SKEY, 76
SKEYID, 75
SKEYSEED, 40, 54, 75
slave priority, 370
smart defaults, 106, 168–169
pre-shared-key authentication with smart defaults, 189–194
smart DPD, auto-reconnect, FlexVPN server, 311–313
SNMP (Simple Network Management Protocol), 419–420
IKE trap commands, 427
with IKEv2, VPN tunnel establishment, 425–427
with IPsec, data encryption, 437–439
trap commands, 438–439
versions, 419–420
VRF-aware SNMP, 420

SNMP agent, 419
SNMP manager, 419
snmp-server enable traps, 425
snmp-server enable traps ike tunnel start, 425
snmp-server enable traps ike tunnel stop, 426
snmp-server enable traps IPsec tunnel start, 437, 438
snmp-server enable traps IPsec tunnel stop, 439
snmp-server enable traps snmp linkdown linkup, 439–440
snmp-server enable traps snmp linkup linkdown, 424
soft migration, transitioning from IKEv1 to IKEv2, 549–559
software, considerations when moving to IKEv2, 543
SPD (Security Policy Database), 5–6
SPI (Security Parameter Index), 3, 5
IKEv2 exchange, 34–35
Split-DNS
attributes, configuring, 341
FlexVPN client, 338–340
components of, 340–343
spoke configuration, FlexVPN
branch 1 configuration, 392–394
branch 2 configuration, 394–395
hub configuration, 395–397
verification at branch 1, 397–399
verification at branch 2, 399–400
verification on hub, 401–404
static P2P tunnel interfaces, 214–216
FlexVPN client, 334
static routing, 496
static tunnel interfaces, 90
static VTI (sVTI), 92
subject information access (SIA), 202
sub-modes
IKEv2 policies, 119
IKEv2 proposals, 108
subnets, remote subnets, FlexVPN, 264, 265–266
sub-policy-in, 304
sub-policy-out, 304
sub-qos-policy-in, 304
sub-qos-policy-out, 304

sVTI (static VTI), 92
SXP (Security group tag exchange), 179
symmetric cryptography, IPsec VPNs, 7–8
symmetric keys, 132
syslog, 421
IKEv2, 428–429
syslog messages, troubleshooting, 447

T

TCAM (ternary content-addressable memory), 87
test aaa command, 481
TFC (Traffic Flow Confidentiality), 4–5, 504
ESP (Encapsulating Security Payload) version 3, 20
timeout keyword, 484
timeout option, FlexVPN server, 275, 278
TLS (transport layer security), EAP methods, 272
tools for troubleshooting, 446–447
event-trace monitoring, 447–449
show commands, 447
syslog messages, 447
topologies, migration strategies, 561
hub-and-spoke topology, 562–564
site-to-site, 561–562
TP (trustpoints), 148, 163–164, 195
configuring, 476
selecting, 476–477
track mode, tunnel, FlexVPN client, 350
track-based tunnel activation, FlexVPN backup tunnels, 414–415
tracking
FlexVPN client, 356
EEM (embedded event manager), 356–358
lists of objects, with boolean expressions, 350–352
traffic
multicast traffic, 85–86
non-IP protocols, 86
routing, 88–90
Traffic Flow Confidentiality. *See* TFC (Traffic Flow Confidentiality), 504
traffic selectors, 4, 50–52, 74
GRE, 51–52
IPsec SA, 51

Transforms, 29–30
transitioning from IKEv1 to IKEv2, 548
hard migration, 548–549
soft migration, 549–559
transport mode
FlexVPN, 219–221
GRE over IPsec, 83–84
IPsec, 20–21
traps
IPsec SNMP trap, 437
SNMP IPsec trap commands, 438–439
troubleshooting
debugging, 449
conditional debugging, 456–457
IKEv2, 449–453
IPsec, 488–491
IPsec debugging, 453
KMI (Key Management Interface), 453–455
PKI (Public Key Infrastructure), 456
FlexVPN client
debugging, 360
show commands, 358–359
IKE_AUTH, 464
ECDSA (Elliptic-Curve Digital Signature Algorithm) authentication, 465–468
RSA authentication, 465–468
IKEv2, diagnose error, 460–461
IKEv2 load balancing, 374–375
IP connectivity, 457–460
IPsec VPN methodology, 446
IKE_SA_INIT exchange, 461–464
tools, 446–447
event-trace monitoring, 447–449
show commands, 447
syslog messages, 447
VPN tunnel establishment, 460
trustpoints (TP), 148, 163–164, 195
configuring, 476
selecting, 476–477
tunnel destination, 88, 215, 218
FlexVPN client, 349

tunnel destination dynamic, 349, 412
tunnel destination peer-address, 90
tunnel encapsulation modes, 215
auto detection, FlexVPN server, 297–298
FlexVPN, 219–221
tunnel endpoints, 88
tunnel initiation, FlexVPN client, 350
automatic mode, 350
manual mode, 350
track mode, 350
tunnel interface, 79, 229
FlexVPN client, 347–348
tunnel mode, 87–88, 215, 218
AH (Authentication Header), 3
ESP (Encapsulating Security Payload), 3
GRE over IPsec, 84–85
IPsec, 21
tunnel mode auto, FlexVPN, 391–392
tunnel mode gre, 214–215, 217
tunnel mode gre ip, 94, 214, 216
tunnel mode IPSEC, 298
tunnel mode IPsec, 214–215, 217
tunnel mode IPsec ipv4, 96
tunnel mode IPsec ipv4 v6-overlay, 97
tunnel mode IPsec ipv6, 96
tunnel mode IPsec ipv6 v4-overlay, 97–98
tunnel path-mtu-discovery, 499, 534–535
Tunnel Pivot, 544
tunnel protection, 80, 94–95, 139, 216, 218
versus crypto maps, 80–81
IPsec parameters, 167
tunnel protection command, 167–168
tunnel protection IPsec, 123
tunnel source, 88, 218
FlexVPN client, 348–349
tunnel source dynamic, 348, 408
tunnel vrf name, 102
Tunnel-Password, 285
tunnels

FlexVPN backup tunnels, track-based tunnel activation, 414–415
fragmentation, 531
GRE (generic routing encapsulation), 532–533
GRE over IPsec, 534
IPsec only (VTI), 531–532
PMTUD (path MTU discovery), 534–535
type tunnel, 91

U

UDP (User Datagram Protocol), 25
uniform resource identifier (URI), 202
updating
session ACL with RADIUS CoA, 307–309
session QoS policies, RADIUS change-of-authorization (CoA), 305–307
URI (uniform resource identifier), 202
user authentication, AnyConnect-EAP, 315, 316–318
user authorization
FlexVPN, 235–237
precedence, 247–249
FlexVPN client, 386
branch 1 configuration, 386
branch 2 configuration, 387
User Datagram Protocol (UDP), 25

V

value proposition, FlexVPN, 213
verbose debugging, 181
verification
FlexVPN client, 409–410, 412–413
NAT (Network Address Translation), 405–407
FlexVPN spoke
branch 1, 397–399
branch 2 configuration, 399–400
hub configuration, 401–404
migration, 559–561
VersionIDofSecret, 36–37
VFRF (Front-door VRF), 118
virtual access cloning, examples, 295–297
virtual access configurations, FlexVPN server
deriving from AAA authorization, 293–294

deriving from incoming sessions, 294
deriving from virtual templates, 291–293
virtual access interfaces, 217
virtual IPsec interfaces, 85–86
Virtual Routing and Forwarding. *See* VRF (Virtual Routing and Forwarding)
virtual template interface
configuring, 216–219
FlexVPN server, virtual access configurations, 291–293
IKEv2 profiles, 153
Virtual Tunnel Interface (VTI), 87–88
virtual-access interface, 290–291
virtual-template 1 mode auto, 396
virtual-template interfaces, FlexVPN feature, 91
VPN gateways, considerations when moving to IKEv2, 543
VPN peers, 2
VPN technology, considerations when moving to IKEv2, 540–541
VPN tunnel establishment, 425, 460
Cisco IPsec flow monitor MIB, 425
SNMP with IKEv2, 425–427
VRF (Virtual Routing and Forwarding), 81, 118
IPsec, 99–101
VRF aware, 101
GRE (generic routing encapsulation), 101–102
GRE over IPsec, 102–103
IPsec, 101–102
vrf definition, 100
vrf forwarding, 100, 216, 218, 486
vrf forwarding name, 102
VRF-aware SNMP, 420
VTI (Virtual Tunnel Interface), 87–88
fragmentation, 531–532

W

wildcard keys, 130
window size, IKEv2, 158
WINS (Windows Internet Naming Service), FlexVPN client, 343–344
worst case maximum overhead, 514–515

X-Y-Z

XAUTH (Extended Authentication within IKE), 69

XML
Aggregate XML, 315
AnyConnect-EAP XML messages, 322–324
XML configurations, AnyConnect, 282–283



REGISTER YOUR PRODUCT at CiscoPress.com/register

Access Additional Benefits and SAVE 35% on Your Next Purchase

- Download available product updates.
- Access bonus material when applicable.
- Receive exclusive offers on new editions and related products. (Just check the box to hear from us when setting up your account.)
- Get a coupon for 35% for your next purchase, valid for 30 days. Your code will be available in your Cisco Press cart. (You will also find it in the Manage Codes section of your account page.)

Registration benefits vary by product. Benefits will be listed on your account page under Registered Products.

CiscoPress.com – Learning Solutions for Self-Paced Study, Enterprise, and the Classroom
Cisco Press is the Cisco Systems authorized book publisher of Cisco networking technology, Cisco certification self-study, and Cisco Networking Academy Program materials.

At CiscoPress.com you can

- Shop our books, eBooks, software, and video training.
- Take advantage of our special offers and promotions (ciscopress.com/promotions).
- Sign up for special offers and content newsletters (ciscopress.com/newsletters).
- Read free articles, exam profiles, and blogs by information technology experts.
- Access thousands of free chapters and video lessons.

Connect with Cisco Press – Visit CiscoPress.com/community
Learn about Cisco Press community events and programs.

Cisco Press

Exclusive Offer – 40% OFF

Cisco Press Video Training

livelessons

ciscopress.com/video

Use coupon code CPVIDEO40 during checkout.



Video Instruction from Technology Experts



Advance Your Skills

Get started with fundamentals, become an expert, or get certified.



Train Anywhere

Train anywhere, at your own pace, on any device.



Learn

Learn from trusted author trainers published by Cisco Press.

Try Our Popular Video Training for FREE!

ciscopress.com/video

Explore hundreds of **FREE** video lessons from our growing library of Complete Video Courses, LiveLessons, networking talks, and workshops.

Cisco Press ciscopress.com/video

ALWAYS LEARNING PEARSON

Connect, Engage, Collaborate

The Award Winning Cisco Support Community

Attend and Participate in Events

Ask the Experts
Live Webcasts

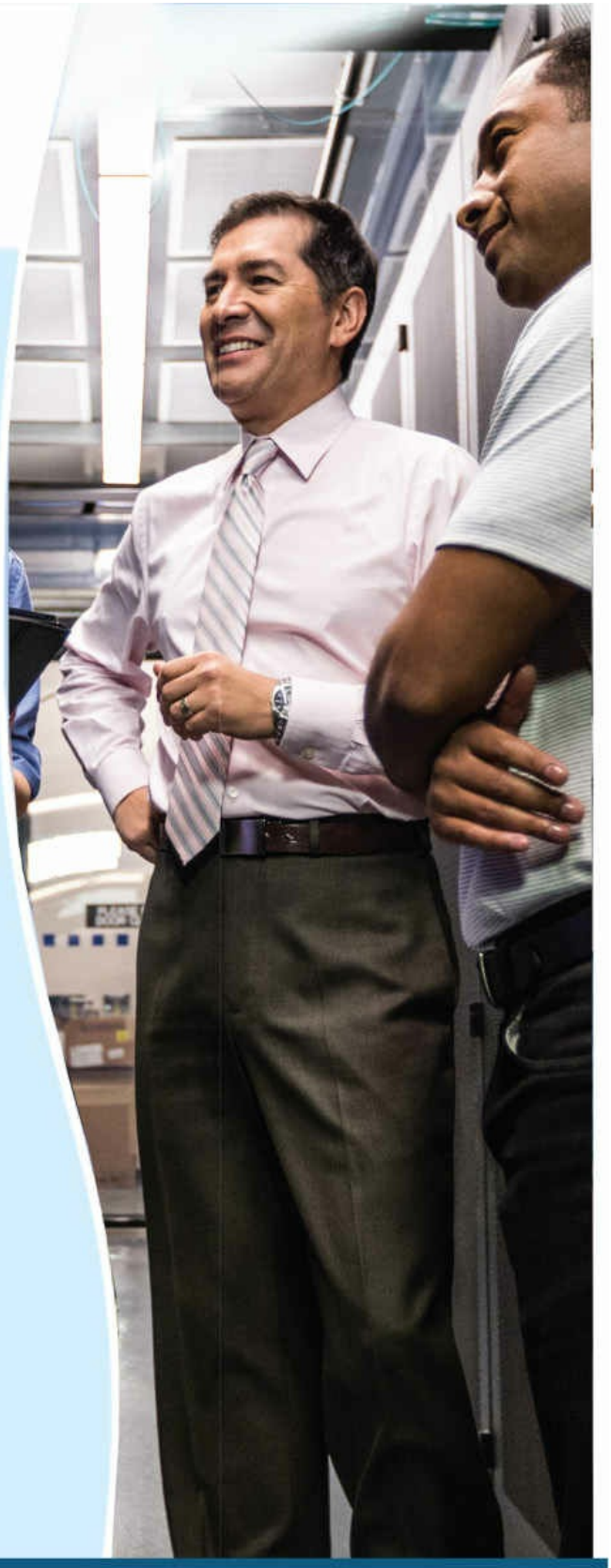
Knowledge Sharing

Documents
Blogs
Videos

Top Contributor Programs

Cisco Designated VIP
Hall of Fame
Spotlight Awards

Multi-Language Support



https://supportforums.cisco.com

Code Snippets

```
    Type Payload: Key Exchange (34)
       Next payload: Nonce (40)
        0... .... = Critical Bit: Not Critical
        Payload length: 104
        DH Group #: Default 768-bit MODP group (1)
        Key Exchange Data:
dcfae25205d1afcbc1b9d9a77464ad375229a747fc48e56621608e7533d617ce2a4b4f54ff3702f
0632c17ac7e8fa8ee1c52e4d0825bd5ff1d26e0197b106dae63bff8183fab5fcf9fe2bbe5ab3fbb
a37f26be4cda595139e894fb01d5d6f6c5
```

```
Type Payload: Security Association (33)
                    Next payload: Traffic Selector - Initiator (44)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 144
                    Type Payload: Proposal (2) # 1
                        Next payload: Proposal (2)
                        0... .... = Critical Bit: Not Critical
                        Payload length: 40
                        Proposal number: 1
                        Protocol ID: ESP (3)
                        SPI Size: 4
                        Proposal transforms: 3
                        SPI: cf1346e0
```

```
Type Payload: Key Exchange (34)
        Next payload: Nonce (40)
        0... .... = Critical Bit: Not Critical
        Payload length: 200
        DH Group #: 1536 bit MODP group (5)
        Key Exchange Data: 973d1a029804cb52720701ce384322686c4f939aa78727e5...
```

```
Type Payload: Security Association (33)
    Next payload: Key Exchange (34)
    0... .... = Critical Bit: Not Critical
    Payload length: 136
    Type Payload: Proposal (2) # 1
        Next payload: NONE / No Next Payload  (0)
        0... .... = Critical Bit: Not Critical
        Payload length: 132
        Proposal number: 1
        Protocol ID: IKE (1)
        SPI Size: 0
        Proposal transforms: 14
        Type Payload: Transform (3)
            Next payload: Transform (3)
            0... .... = Critical Bit: Not Critical
            Payload length: 12
            Transform Type: Encryption Algorithm (ENCR) (1)
            Transform ID (ENCR): ENCR_AES_CBC (12)
            Transform IKE2 Attribute Type (t=14,l=2) Key-Length : 256
                1... .... .... .... = Transform IKE2 Format: Type/Value
                Transform IKE2 Attribute Type: Key-Length (14)
```

```
        Value: 0100
        Key Length: 256
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 12
    Transform Type: Encryption Algorithm (ENCR) (1)
    Transform ID (ENCR): ENCR_AES_CBC (12)
    Transform IKE2 Attribute Type (t=14,l=2) Key-Length : 192
        1... .... .... .... = Transform IKE2 Format: Type/Value
        Transform IKE2 Attribute Type: Key-Length (14)
        Value: 00c0
        Key Length: 192
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 12
    Transform Type: Encryption Algorithm (ENCR) (1)
    Transform ID (ENCR): ENCR_AES_CBC (12)
    Transform IKE2 Attribute Type (t=14,l=2) Key-Length : 128
        1... .... .... .... = Transform IKE2 Format: Type/Value
        Transform IKE2 Attribute Type: Key-Length (14)
```

```
        Value: 0080
        Key Length: 128
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 8
    Transform Type: Pseudo-random Function (PRF) (2)
    Transform ID (PRF): PRF_HMAC_SHA2_512 (7)
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 8
    Transform Type: Pseudo-random Function (PRF) (2)
    Transform ID (PRF): PRF_HMAC_SHA2_384 (6)
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 8
    Transform Type: Pseudo-random Function (PRF) (2)
    Transform ID (PRF): PRF_HMAC_SHA2_256 (5)
Type Payload: Transform (3)
```

```
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 8
    Transform Type: Pseudo-random Function (PRF) (2)
    Transform ID (PRF): PRF_HMAC_SHA1 (2)
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 8
    Transform Type: Pseudo-random Function (PRF) (2)
    Transform ID (PRF): PRF_HMAC_MD5 (1)
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 8
    Transform Type: Integrity Algorithm (INTEG) (3)
    Transform ID (INTEG): AUTH_HMAC_SHA2_512_256 (14)
```

```
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 8
    Transform Type: Integrity Algorithm (INTEG) (3)
    Transform ID (INTEG): AUTH_HMAC_SHA2_384_192 (13)
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 8
    Transform Type: Integrity Algorithm (INTEG) (3)
    Transform ID (INTEG): AUTH_HMAC_SHA2_256_128 (12)
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 8
    Transform Type: Integrity Algorithm (INTEG) (3)
    Transform ID (INTEG): AUTH_HMAC_SHA1_96 (2)
Type Payload: Transform (3)
    Next payload: Transform (3)
    0... .... = Critical Bit: Not Critical
    Payload length: 8
    Transform Type: Integrity Algorithm (INTEG) (3)
```

```
            Transform ID (INTEG): AUTH_HMAC_MD5_96 (1)
        Type Payload: Transform (3)
            Next payload: Transform (3)
            0... .... = Critical Bit: Not Critical
            Payload length: 8
            Transform Type: Diffie-Hellman Group (D-H) (4)
            Transform ID (D-H): 1536 bit MODP group (5)
Type Payload: Transform (3)
            Next payload: NONE / No Next Payload  (0)
            0... .... = Critical Bit: Not Critical
            Payload length: 8
            Transform Type: Diffie-Hellman Group (D-H) (4)
            Transform ID (D-H): 1024 bit MODP group (2)
```

```
IKEv2 proposal: default
     Encryption : AES-CBC-256 AES-CBC-192 AES-CBC-128
     Integrity  : SHA512 SHA384 SHA256 SHA96 MD596
     PRF        : SHA512 SHA384 SHA256 SHA1 MD5
     DH Group   : DH_GROUP_1536_MODP/Group 5 DH_GROUP_1024_MODP/Group 2
```

```
Internet Security Association and Key Management Protocol
    Initiator SPI: 7803611e4b2ff28c
    Responder SPI: 0000000000000000
    Next payload: Security Association (33)
    Version: 2.0
        0010 .... = MjVer: 0x02
        .... 0000 = MnVer: 0x00
    Exchange type: IKE_SA_INIT (34)
    Flags: 0x08 (Initiator, No higher version, Request)
        .... 1... = Initiator: Initiator
        ...0 .... = Version: No higher version
        ..0. .... = Response: Request
```

Internet Security Association and Key Management Protocol

Initiator SPI: 7803611e4b2ff28c

Responder SPI: 5d1b50735fe39ffc

Next payload: Security Association (33)

Version: 2.0

0010 .... = MjVer: 0x02

.... 0000 = MnVer: 0x00

Exchange type: IKE_SA_INIT (34)

Flags: 0x20 (Responder, No higher version, Response)

.... 0... = Initiator: Responder

...0 .... = Version: No higher version

..1. .... = Response: Response

```
Type Payload: Nonce (40)
        Next payload: Notify (41)
        0... .... = Critical Bit: Not Critical
        Payload length: 36
        Nonce DATA: 559292633baf45cf133f7ceebf963608c2f481bdc715ac67...
```

Cookie = <*VersionIDofSecret*> | Hash (Ni | IPi | SPIi | <*secret*>)

```
Type Payload: Certificate Request (38)
        Next payload: Notify (41)
        0... .... = Critical Bit: Not Critical
        Payload length: 105
        Certificate Type: Hash and URL of X.509 certificate (12)
        Certificate Authority Data: e27f7bd877d5df9e0a3f9eb4cb0e2ea9efdb6977
        Certificate Authority Data: 3c76492ca10b9f850e2fd6fbae0d362daa2b9c01
        Certificate Authority Data: d0c52226ab4f4660ecae0591c7dc5ad1b047f76c
        Certificate Authority Data: 27f3c8151e6e9a020916ad2ba089605fda7b2faa
        Certificate Authority Data: a6031d7fcabdb29140c6cb82361f6b988fddbc29
```

```
Type Payload: Notify (41)
        Next payload: NONE / No Next Payload  (0)
        0... .... = Critical Bit: Not Critical
        Payload length: 8
        Protocol ID: IKE (1)
        SPI Size: 0
        Notify Message Type: HTTP_CERT_LOOKUP_SUPPORTED (16392)
```

```
Type Payload: Encrypted and Authenticated (46)
    Next payload: Security Association (33)
    0... .... = Critical Bit: Not Critical
    Payload length: 160
    Initialization Vector: 36a1dd71fbeaaa683c2254e8530aa781 (16 bytes)
    Encrypted Data (128 bytes)
        Padding (11 bytes)
        Pad Length: 11
    Integrity Checksum Data: ef3e7c44eff24d09902ebcfc (12 bytes)[correct]
```

User Datagram Protocol, Src Port: 500 (500), Dst Port: 500 (500)

Internet Security Association and Key Management Protocol

Initiator SPI: a3ae32c968f6ccf0

Responder SPI: fc1c21ef24ef360c

Next payload: Encrypted and Authenticated (46)

Version: 2.0

Exchange type: IKE_AUTH (35)

Flags: 0x08 (Initiator, No higher version, Request)

Message ID: 0x00000001

Length: 572

Type Payload: Encrypted and Authenticated (46)

```
InitiatorSignedOctets = RealMessage1 | NonceRData | MACedIDForI    GenIKEHDR = [
four octets 0 if using port 4500 ] | RealIKEHDR

RealIKEHDR =  SPIi | SPIr |  . . . | Length

RealMessage1 = RealIKEHDR | RestOfMessage1

NonceRPayload = PayloadHeader | NonceRData

InitiatorIDPayload = PayloadHeader | RestOfInitIDPayload

RestOfInitIDPayload = IDType | RESERVED | InitIDData

MACedIDForI = prf(SK_pi, RestOfInitIDPayload)
```

```
ResponderSignedOctets = RealMessage2 | NonceIData | MACedIDForR    GenIKEHDR = [
four octets 0 if using port 4500 ] | RealIKEHDR

RealIKEHDR =  SPIi | SPIr |  . . . | Length

RealMessage2 = RealIKEHDR | RestOfMessage2

NonceIPayload = PayloadHeader | NonceIData

ResponderIDPayload = PayloadHeader | RestOfRespIDPayload    RestOfRespIDPayload =
IDType | RESERVED | RespIDData

MACedIDForR = prf(SK_pr, RestOfRespIDPayload)
```

AUTH = prf(prf(*Shared Secret*, "Key Pad for IKEv2"), <*InitiatorSignedOctets*>)

```
AUTH = prf( prf(Shared Secret, "Key Pad for IKEv2"), <ResponderSignedOctets>)
```

```
Type Payload: Traffic Selector - Initiator (44) # 1
                    Next payload: Traffic Selector - Responder (45)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 24
                    Number of Traffic Selector: 1
                    Traffic Selector Type: TS_IPV4_ADDR_RANGE (7)
                    Protocol ID: Unused
                    Selector Length: 16
                    Start Port: 0
                    End Port: 65535
                    Starting Addr: 192.168.2.0 (192.168.2.0)
                    Ending Addr: 192.168.2.255 (192.168.2.255)

Type Payload: Traffic Selector - Responder (45) # 1
                    Next payload: NONE / No Next Payload  (0)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 24
                    Number of Traffic Selector: 1
                    Traffic Selector Type: TS_IPV4_ADDR_RANGE (7)
                    Protocol ID: Unused
                    Selector Length: 16
                    Start Port: 0
                    End Port: 65535
                    Starting Addr: 192.168.1.0 (192.168.1.0)
                    Ending Addr: 192.168.1.255 (192.168.1.255)
```

```
Type Payload: Traffic Selector - Initiator (44) # 1
                    Next payload: Traffic Selector - Responder (45)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 24
                    Number of Traffic Selector: 1
                    Traffic Selector Type: TS_IPV6_ADDR_RANGE (8)
                    Protocol ID: 47
                    Selector Length: 40
                    Start Port: 0
                    End Port: 65535
                    Starting Addr: 2003::3 (2003::3)
                    Ending Addr: 2003::3 (2003::3)

Type Payload: Traffic Selector - Responder (45) # 1
                    Next payload: NONE / No Next Payload  (0)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 24
                    Number of Traffic Selector: 1
                    Traffic Selector Type: TS_IPV6_ADDR_RANGE (8)
                    Protocol ID: 47
                    Selector Length: 40
                    Start Port: 0
                    End Port: 65535
                    Starting Addr: 2007::14 (2007::14)
                    Ending Addr: 2007::14 (2007::14)
```

```
Type Payload: Notify (41)
                    Next payload: Identification - Responder (36)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 8
                    Protocol ID: RESERVED (0)
                    SPI Size: 0
                    Notify Message Type: INITIAL_CONTACT (16384)
```

```
User Datagram Protocol, Src Port: 500 (500), Dst Port: 500 (500)
    Source Port: 500 (500)
    Destination Port: 500 (500)
    Length: 408
    Checksum: 0xc089
Internet Security Association and Key Management Protocol
    Initiator SPI: 9ff10b12e411fc0f
    Responder SPI: 0000000000000000
    Next payload: Security Association (33)
    Version: 2.0
        0010 .... = MjVer: 0x02
        .... 0000 = MnVer: 0x00
    Exchange type: IKE_SA_INIT (34)
    Flags: 0x08 (Initiator, No higher version, Request)
        .... 1... = Initiator: Initiator
        ...0 .... = Version: No higher version
        ..0. .... = Response: Request
    Message ID: 0x00000000
    Length: 400
```

```
Internet Security Association and Key Management Protocol
    Initiator SPI: 1739c43d75f18743
    Responder SPI: 1f9bde873d9281f5
    Next payload: Encrypted and Authenticated (46)
    Version: 2.0
    Exchange type: INFORMATIONAL (37)
    Flags: 0x08 (Initiator, No higher version, Request)
    Message ID: 0x00000005
    Length: 76
    Type Payload: Encrypted and Authenticated (46)
        Next payload: Delete (42)
        0... .... = Critical Bit: Not Critical
        Payload length: 48
        Initialization Vector: 459cee48eae9966ac0be082687da717f (16 bytes)
        Encrypted Data (16 bytes)
        Decrypted Data (16 bytes)
            Contained Data (12 bytes)
                Type Payload: Delete (42)
                    Next payload: NONE / No Next Payload  (0)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 12
                    Protocol ID: ESP (3)
                    SPI Size: 4
                    Port: 1
                    Delete SPI: cb989cce
            Padding (3 bytes)
            Pad Length: 3
        Integrity Checksum Data: ccb4a8512ff5ef126611b4b3 (12 bytes)[correct]
```

```
no crypto ipsec nat-transparency udp-encapsulation
```

```
Router#show running-config interface Tunnel0
interface Tunnel0
 ip address 10.0.0.1 255.255.255.252
 tunnel source Ethernet0/0
 tunnel destination 209.165.200.225
end

Router#show running-config | include ip route
ip route 0.0.0.0 0.0.0.0 209.165.201.2
ip route 10.10.210.0 255.255.255.0 10.0.0.2

Router#show ip route 10.10.210.99
Routing entry for 10.10.210.0/24
  Known via "static", distance 1, metric 0
  Routing Descriptor Blocks:
  * 10.0.0.2
      Route metric is 0, traffic share count is 1

Router#show ip route 10.0.0.2
Routing entry for 10.0.0.0/30
  Known via "connected", distance 0, metric 0 (connected, via interface)
  Routing Descriptor Blocks:
  * directly connected, via tunnel0
      Route metric is 0; traffic share count is 1
```

```
interface Tunnel0
 ip address 10.0.0.1 255.255.255.252
 tunnel source Ethernet0/0
 tunnel mode ipsec ipv4
 tunnel destination 209.165.200.225
 tunnel protection ipsec profile default
```

```
interface Loopback1
 ip address 10.0.0.1 255.255.255.255

interface Virtual-Template1 type tunnel
 ip unnumbered Loopback1
 tunnel mode ipsec ipv4
 tunnel protection ipsec profile default
```

```
router#show derived-config interface virtual-access 1
Building configuration…

Derived configuration : 225 bytes

interface Virtual-Access1
 ip unnumbered Loopback1
 tunnel source 209.165.200.225
 tunnel mode ipsec ipv4
 tunnel destination 209.165.201.1
 tunnel protection ipsec profile default
 no tunnel protection ipsec initiate
end
```

```
interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 ip nhrp network-id 1
 ip nhrp nhs 10.0.0.251 nmba 209.165.200.225 multicast
 tunnel source Ethernet0/0
 tunnel mode gre multipoint
 tunnel protection ipsec profile default
```

```
crypto ipsec profile default
 set ikev2-profile default

interface Tunnel0
 ip address 10.0.0.1 255.255.255.252
 tunnel source Ethernet0/0
 tunnel destination 209.165.200.225
 tunnel protection ipsec profile default
```

```
router#show crypto sockets

Number of Crypto Socket connections 1

   Tu0 Peers (local/remote): 209.165.201.1/209.165.200.225
       Local Ident  (addr/mask/port/prot): (209.165.201.1/255.255.255.255/0/47)
       Remote Ident (addr/mask/port/prot): (209.165.200.225/255.255.255.255/0/47)
       IPSec Profile: "default"
       Socket State: Open
       Client: "TUNNEL SEC" (Client State: Active)

Crypto Sockets in Listen state:
Client: "TUNNEL SEC" Profile: "default" Map-name: "Tunnel0-head-0"
```

```
router#show crypto session
Crypto session current status

Interface: Tunnel0
Session status: UP-ACTIVE
Peer: 209.165.200.225 port 500
  IKEv2 SA: local 209.165.201.1/500 remote 209.165.200.225/500 Active
  IPSEC FLOW: permit 47 host 209.165.201.1 host 209.165.200.225
        Active SAs: 2, origin: crypto-map
```

```
interface Tunnel0
    ip address 192.168.1.1 255.255.255.0
    tunnel source GigabitEthernet0/0
    tunnel mode ipsec ipv6 v4-overlay
    tunnel destination 2001:DB8::2
    tunnel protection ipsec profile default
```

```
Router#show crypto session
Crypto session current status
Interface: Tunnel0
Profile: default
Session status: UP-ACTIVE
Peer: 2001:DB8::2 port 4500
  Session ID: 1
  IKEv2 SA: local 2001:DB8::1/500 remote 2001:DB8::2/500 Active
  IPSEC FLOW: permit ip 0.0.0.0/0.0.0.0 0.0.0.0/0.0.0.0
        Active SAs: 2, origin: crypto-map
```

tunnel mode ipsec ipv4 v6-overlay

```
interface Tunnel0
    ipv6 address 2001:DB8::1/112
    ipv6 enable
    tunnel source GigabitEthernet0/0
    tunnel mode ipsec ipv4 v6-overlay
    tunnel destination 192.168.1.2
    tunnel protection ipsec profile default
```

```
Router#show crypto session
Crypto session current status
Interface: Tunnel0
Profile: default
Session status: UP-ACTIVE
Peer: 192.168.1.2 port 4500
  Session ID: 1
  IKEv2 SA: local 192.168.1.1/500 remote 192.168.1.2/500 Active
  IPSEC FLOW: permit ipv6 ::/0 ::/0
        Active SAs: 2, origin: crypto-map
```

```
vrf definition blue
 address-family ipv4
 exit-address-family

vrf definition red
 address-family ipv4
 exit-address-family

interface Ethernet0/1
 vrf forwarding red
 ip address 172.16.10.1 255.255.255.0

interface Ethernet0/2
 vrf forwarding red
 ip address 172.16.20.1 255.255.255.0

interface Ethernet1/1
 vrf forwarding blue
 ip address 172.16.10.1 255.255.255.0
```

```
interface Ethernet1/2
 vrf forwarding blue
 ip address 172.16.30.1 255.255.255.0

s1#show ip route vrf red
Routing Table: red
[...]
      172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks
C        172.16.10.0/24 is directly connected, Ethernet0/1
L        172.16.10.1/32 is directly connected, Ethernet0/1
C        172.16.20.0/24 is directly connected, Ethernet0/2
L        172.16.20.1/32 is directly connected, Ethernet0/2

s1#show ip route vrf blue
Routing Table: blue
      172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks
C        172.16.10.0/24 is directly connected, Ethernet1/1
L        172.16.10.1/32 is directly connected, Ethernet1/1
C        172.16.30.0/24 is directly connected, Ethernet1/2
L        172.16.30.1/32 is directly connected, Ethernet1/2
```

crypto ikev2 proposal *proposal-name*

```
Router(config)#crypto ikev2 proposal ikev2_proposal
IKEv2 proposal MUST either have a set of an encryption algorithm other than
aes-gcm, an integrity algorithm and a DH group configured or
 encryption algorithm aes-gcm, a prf algorithm and a DH group configured
Router(config-ikev2-proposal)#
```

```
Router(config-ikev2-proposal)#?
IKEv2 Proposal commands:
  encryption  Set encryption algorithm(s) for proposal
  exit        Exit from IKEv2 proposal configuration mode
  group       Set the Diffie-Hellman group(s)
  integrity   Set integrity hash algorithm(s) for proposal
  no          Negate a command or set its defaults
  prf         Set prf algorithm(s) for proposal
```

```
Router#show run | begin ikev2 proposal
crypto ikev2 proposal ikev2_proposal
 ! Proposal Incomplete(MUST have atleast an encryption algorithm, an integrity
algorithm and a dh group configured)
```

```
Router#show crypto ikev2 proposal
 IKEv2 proposal: default
     Encryption : AES-CBC-256 AES-CBC-192 AES-CBC-128
     Integrity  : SHA512 SHA384 SHA256 SHA96 MD596
     PRF        : SHA512 SHA384 SHA256 SHA1 MD5
     DH Group   : DH_GROUP_1536_MODP/Group 5 DH_GROUP_1024_MODP/Group 2
 IKEv2 proposal: ikev2_proposal
     Encryption : AES-GCM-256 AES-GCM-128
     Integrity  : none
     PRF        : SHA256 SHA384
     DH Group   : DH_GROUP_384_ECP/Group 20 DH_GROUP_521_ECP/Group 21
```

```
Router(config)#crypto ikev2 policy ikev2_policy
Router(config-ikev2-policy)#proposal ikev2_proposal
```

```
Router(config)#no crypto ikev2 proposal ikev2_proposal
% Cannot remove as proposal is in use.
Router(config)#crypto ikev2 policy ikev2_policy
Router(config-ikev2-policy)#no proposal ikev2_proposal
Router(config-ikev2-policy)#exit
Router(config)#no crypto ikev2 proposal ikev2_proposal
Router(config)#
```

```
Router(config)#crypto ikev2 proposal ikev2_proposal
Router(config-ikev2-proposal)#encryption aes-cbc-256 ?
  3des          3DES
  aes-cbc-128   AES-CBC-128
  aes-cbc-192   AES-CBC-192
  des           DES
  <cr>

Router(config-ikev2-proposal)#encryption aes-cbc-256
Router(config-ikev2-proposal)#integrity sha512
Router(config-ikev2-proposal)#prf sha512
Router(config-ikev2-proposal)#group 16
```

```
Router#show run | begin ikev2 proposal
crypto ikev2 proposal ikev2_proposal
 encryption aes-cbc-256
 integrity sha512
 group 16

Router#show crypto ikev2 proposal ikev2_proposal
 IKEv2 proposal: ikev2_proposal
      Encryption : AES-CBC-256
      Integrity  : SHA512
      PRF        : SHA512
      DH Group   : DH_GROUP_4096_MODP/Group 16
```

```
Router(config)#crypto ikev2 proposal ikev2_proposal
Router(config-ikev2-proposal)#encryption aes-gcm-128 ?
  aes-gcm-256  Combined-mode,256 bit key,16 byte ICV(Authentication Tag)
  <cr>

Router(config-ikev2-proposal)#encryption aes-gcm-256 aes-gcm-128 ?
  <cr>

Router(config-ikev2-proposal)#integrity sha512
 Warning!! Integrity algorithms are not allowed with AES_GCM
Router(config-ikev2-proposal)#group 21 20
Router(config-ikev2-proposal)#prf sha512 sha384
Router(config-ikev2-proposal)#
```

```
Router#show run | begin ikev2 proposal
crypto ikev2 proposal ikev2_proposal
 encryption aes-gcm-256 aes-gcm-128
 prf sha512 sha384
 group 21 20

Router#show crypto ikev2 proposal ikev2_proposal
IKEv2 proposal: ikev2_proposal
      Encryption : AES-GCM-256 AES-GCM-128
      Integrity  : none
      PRF        : SHA512 SHA384
      DH Group   : DH_GROUP_521_ECP/Group 21 DH_GROUP_384_ECP/Group 20
```

```
Router(config-ikev2-proposal)#encryption ?
  3des          3DES
  aes-cbc-128   AES-CBC-128
  aes-cbc-192   AES-CBC-192
  aes-cbc-256   AES-CBC-256
  aes-gcm-128   Combined-mode,128 bit key,16 byte ICV(Authentication Tag)
  aes-gcm-256   Combined-mode,256 bit key,16 byte ICV(Authentication Tag)
  des           DES
```

```
Router(config-ikev2-proposal)#integrity sha384 sha256 sha1
```

```
Router(config-ikev2-proposal)#group 20 21 19 16
```

```
Router(config-ikev2-proposal)#prf sha256 sha1 md5
```

```
Router#show crypto ikev2 proposal default
 IKEv2 proposal: default
      Encryption : AES-CBC-256 AES-CBC-192 AES-CBC-128
      Integrity  : SHA512 SHA384 SHA256 SHA96 MD596
      PRF        : SHA512 SHA384 SHA256 SHA1 MD5
      DH Group   : DH_GROUP_1536_MODP/Group 5 DH_GROUP_1024_MODP/Group 2

Router#show crypto ikev2 policy default
 IKEv2 policy : default
      Match fvrf : any
      Match address local : any
      Proposal    : default
```

```
Router(config)#no crypto ikev2 proposal default
% Cannot remove as proposal is in use.

Router(config)#no crypto ikev2 policy default
Router(config)#no crypto ikev2 proposal default

Router#show run | include ikev2 proposal default
no crypto ikev2 proposal default

Router#show crypto ikev2 proposal default
 IKEv2 proposal: default Disabled
```

```
Router(config)#default crypto ikev2 proposal

Router#show run | include ikev2 proposal default
Router#show crypto ikev2 proposal default
 IKEv2 proposal: default
     Encryption : AES-CBC-256 AES-CBC-192 AES-CBC-128
     Integrity  : SHA512 SHA384 SHA256 SHA96 MD596
     PRF        : SHA512 SHA384 SHA256 SHA1 MD5
     DH Group   : DH_GROUP_1536_MODP/Group 5 DH_GROUP_1024_MODP/Group 2
```

```
Router(config)#crypto ikev2 proposal default
%Warning: This will Modify Default IKEv2 Proposal. Exit if you don't want
```

```
Router(config-ikev2-proposal)#no integrity sha512 sha384 sha256 sha1 md5
Router(config-ikev2-proposal)#encryption aes-gcm-256
Router(config-ikev2-proposal)#prf sha512
Router(config-ikev2-proposal)#group 21
Router(config-ikev2-proposal)#

Router#show run | begin ikev2 proposal default
crypto ikev2 proposal default
 encryption aes-gcm-256
 prf sha512
 group 21

Router#show crypto ikev2 proposal default
 IKEv2 proposal: default
     Encryption : AES-GCM-256
     Integrity  : none
     PRF        : SHA512
     DH Group   : DH_GROUP_521_ECP/Group 21
```

```
Router(config)#default crypto ikev2 proposal
Router#show run | begin ikev2 proposal default
Router#show crypto ikev2 proposal default
 IKEv2 proposal: default
     Encryption : AES-CBC-256 AES-CBC-192 AES-CBC-128
     Integrity  : SHA512 SHA384 SHA256 SHA96 MD596
     PRF        : SHA512 SHA384 SHA256 SHA1 MD5
     DH Group   : DH_GROUP_1536_MODP/Group 5 DH_GROUP_1024_MODP/Group 2
```

crypto ikev2 policy *policy-name*

```
Router(config)#crypto ikev2 policy ikev2_policy
IKEv2 policy MUST have at least one complete proposal attached
Router(config-ikev2-policy)#
```

```
Router(config-ikev2-policy)#?

IKEv2 Policy commands:
  exit       Exit from IKEv2 policy configuration mode
  match      Match values of local fields
  no         Negate a command or set its defaults
  proposal   Specify Proposal
```

```
Router#show run | begin ikev2 policy
crypto ikev2 policy temp-ikev2-policy1
 ! Policy Incomplete(MUST have at least one complete proposal attached)
```

```
Router(config)#crypto ikev2 policy ikev2_policy
Router(config-ikev2-policy)#proposal ikev2_proposal1
% No Proposal exists with the specified name ikev2_proposal1

Router(config)#crypto ikev2 proposal ikev2_proposal1
IKEv2 proposal MUST either have a set of an encryption algorithm other than
aes-gcm, an integrity algorithm and a DH group configured or
 encryption algorithm aes-gcm, a prf algorithm and a DH group configured

Router(config)#crypto ikev2 policy ikev2_policy
Router(config-ikev2-policy)#proposal ikev2_proposal1
Proposal ikev2_proposal1 is not complete to be attached to policy
```

```
Router(config)#crypto ikev2 policy ikev2_policy
Router(config-ikev2-policy)#proposal ikev2_proposal1
Router(config-ikev2-policy)#proposal ikev2_proposal2
```

```
Router(config)#crypto ikev2 policy ikev2_policy
Router(config-ikev2-policy)#match ?
  address  Specify the address to match
  fvrf     Specify fvrf

Router(config-ikev2-policy)#match fvrf ?
  WORD  fvrf name (default:global)
  any   Any fvrf

Router(config-ikev2-policy)#match address ?
  local  Local address

Router(config-ikev2-policy)#match address local ?
  A.B.C.D      IPv4 address
  X:X:X:X::X   IPv6 address
```

```
Router(config-ikev2-policy)#match fvrf mpls
Router(config-ikev2-policy)#match fvrf internet
Router(config-ikev2-policy)#match address local 192.168.1.1
Router(config-ikev2-policy)#match address local 2001::1
```

```
Router#show run | begin ikev2 policy
crypto ikev2 policy ikev2_policy
 match fvrf mpls
 match fvrf internet
 match address local 192.168.1.1
 match address local 2001::1
 proposal ikev2_proposal1
 proposal ikev2_proposal2

Router#show crypto ikev2 policy ikev2_policy
 IKEv2 policy : ikev2_policy
       Match fvrf : mpls
       Match fvrf : internet
       Match address local : 192.168.1.1
       Match address local : 2001::1
       Proposal    : ikev2_proposal1
       Proposal    : ikev2_proposal2
```

```
Router#show crypto ikev2 policy default

 IKEv2 policy : default
       Match fvrf : any
       Match address local : any
       Proposal    : default
```

```
Router(config)#no crypto ikev2 policy default

Router#show run | include ikev2 policy default
no crypto ikev2 policy default

Router#show crypto ikev2 policy default
 IKEv2 policy : default Disabled
```

```
Router(config)#default crypto ikev2 policy
```

```
Router(config)#crypto ikev2 policy default
Router(config-ikev2-policy)#match fvrf mpls
% Specific vrf cannot be configured when fvrf type any is already configured.
Remove any fvrf and then add specific
```

```
Router(config-ikev2-policy)#no match fvrf any
Router(config-ikev2-policy)#match fvrf mpls

Router#show run | begin ikev2 policy default
crypto ikev2 policy default
 match fvrf mpls
 proposal default

Router#show crypto ikev2 policy default
 IKEv2 policy : default
       Match fvrf : mpls
       Match address local : any
       Proposal    : default
```

```
Router(config)#default crypto ikev2 policy
```

```
interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 tunnel source 192.168.1.1
 tunnel mode ipsec ipv4
 tunnel destination 192.168.1.2
 tunnel protection ipsec profile default


KMI: (Session ID: 3) IPSEC key engine sending message KEY_ENG_REQUEST_SAS to
Crypto IKEv2.
KMI: (Session ID: 3) Crypto IKEv2 received message KEY_ENG_REQUEST_SAS from IPSEC
key engine.
IKEv2:Searching Policy with fvrf 0, local address 192.168.1.1
IKEv2:Using the Default Policy for Proposal
IKEv2:Found Policy 'default'
IKEv2:(SESSION ID = 3,SA ID = 1):Generating IKE_SA_INIT message
IKEv2:(SESSION ID = 3,SA ID = 1):Sending Packet [To 192.168.1.2:500/From
192.168.1.1:500/VRF i0:f0]
Initiator SPI : CF2306A9194583C6 - Responder SPI : 0000000000000000 Message id: 0
IKEv2 IKE_SA_INIT Exchange REQUEST
```

```
IKEv2:Searching Policy with fvrf 0, local address 192.168.1.1
IKEv2-ERROR:Failed to initiate sa
```

```
IKEv2:Received Packet [From 192.168.1.1:500/To 192.168.1.2:500/VRF i0:f0]
Initiator SPI : C51E7F53FCA49E64 - Responder SPI : 0000000000000000 Message id: 0
IKEv2 IKE_SA_INIT Exchange REQUEST
Payload contents:
 SA KE N VID VID NOTIFY(NAT_DETECTION_SOURCE_IP)
NOTIFY(NAT_DETECTION_DESTINATION_IP)
IKEv2:(SESSION ID = 850,SA ID = 1):Verify SA init message
IKEv2:(SESSION ID = 850,SA ID = 1):Insert SA
IKEv2:Searching Policy with fvrf 0, local address 192.168.1.2
IKEv2:Using the Default Policy for Proposal
IKEv2:Found Policy 'default'
```

```
IKEv2:Received Packet [From 192.168.1.1:500/To 192.168.1.2:500/VRF i0:f0]
Initiator SPI : 418BD852B092D462 - Responder SPI : 0000000000000000 Message id: 0
IKEv2 IKE_SA_INIT Exchange REQUEST
Payload contents:
 SA KE N VID VID NOTIFY(NAT_DETECTION_SOURCE_IP)
NOTIFY(NAT_DETECTION_DESTINATION_IP)

IKEv2:(SESSION ID = 851,SA ID = 1):Verify SA init message
IKEv2:(SESSION ID = 851,SA ID = 1):Insert SA
IKEv2:Searching Policy with fvrf 0, local address 192.168.1.2
IKEv2-ERROR:(SESSION ID = 851,SA ID = 1):: Failed to locate an item in the
database
IKEv2:(SESSION ID = 851,SA ID = 1):Failed SA init exchange
IKEv2-ERROR:(SESSION ID = 851,SA ID = 1):Initial exchange failed: Initial exchange
failed
IKEv2:(SESSION ID = 851,SA ID = 1):Abort exchange
IKEv2:(SESSION ID = 851,SA ID = 1):Deleting SA
```

```
Router#show crypto ikev2 proposal
 IKEv2 proposal: Proposal1
      Encryption : AES-GCM-128
      Integrity  : none
      PRF        : SHA256
      DH Group   : DH_GROUP_256_ECP/Group 19
 IKEv2 proposal: Proposal2
      Encryption : AES-CBC-128
      Integrity  : SHA96
      PRF        : SHA1
      DH Group   : DH_GROUP_2048_MODP/Group 14


Router#show crypto ikev2 policy
 IKEv2 policy : Policy1
       Match fvrf  : global
       Match address local : 192.168.1.1
       Proposal    : Proposal1

 IKEv2 policy : Policy2
       Match fvrf  : global
       Match address local : 192.168.2.1
       Proposal    : Proposal2
```

```
Router#show crypto ikev2 proposal
 IKEv2 proposal: Proposal1
      Encryption : AES-GCM-128
      Integrity  : none
      PRF        : SHA256
      DH Group   : DH_GROUP_256_ECP/Group 19
 IKEv2 proposal: Proposal2
      Encryption : AES-CBC-128
      Integrity  : SHA96
      PRF        : SHA1
      DH Group   : DH_GROUP_2048_MODP/Group 14

Router#show crypto ikev2 policy
 IKEv2 policy : default
       Match fvrf : any
       Match address local : any
       Proposal    : Proposal1
       Proposal    : Proposal2
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):Generating IKE_SA_INIT message
IKEv2:(SESSION ID = 1,SA ID = 1):IKE Proposal: 1, SPI size: 0 (initial
negotiation),
Num. transforms: 3
   AES-GCM   SHA256   DH_GROUP_256_ECP/Group 19
IKEv2:(SESSION ID = 1,SA ID = 1):IKE Proposal: 2, SPI size: 0 (initial
negotiation),
Num. transforms: 4
   AES-CBC   SHA1   SHA96   DH_GROUP_2048_MODP/Group 14

IKEv2:(SESSION ID = 1,SA ID = 1):Next payload: SA, version: 2.0 Exchange type:
IKE_SA_INIT, flags: INITIATOR Message id: 0, length: 308
Payload contents:
 SA  Next payload: KE, reserved: 0x0, length: 84
  last proposal: 0x2, reserved: 0x0, length: 36
  Proposal: 1, Protocol id: IKE, SPI size: 0, #trans: 3    last transform: 0x3,
reserved: 0x0: length: 12
    type: 1, reserved: 0x0, id: AES-GCM
    last transform: 0x3, reserved: 0x0: length: 8
    type: 2, reserved: 0x0, id: SHA256
    last transform: 0x0, reserved: 0x0: length: 8
    type: 4, reserved: 0x0, id: DH_GROUP_256_ECP/Group 19
  last proposal: 0x0, reserved: 0x0, length: 44
  Proposal: 2, Protocol id: IKE, SPI size: 0, #trans: 4    last transform: 0x3,
reserved: 0x0: length: 12
    type: 1, reserved: 0x0, id: AES-CBC
    last transform: 0x3, reserved: 0x0: length: 8
    type: 2, reserved: 0x0, id: SHA1
    last transform: 0x3, reserved: 0x0: length: 8
    type: 3, reserved: 0x0, id: SHA96
    last transform: 0x0, reserved: 0x0: length: 8
    type: 4, reserved: 0x0, id: DH_GROUP_2048_MODP/Group 14
 KE  Next payload: N, reserved: 0x0, length: 72
    DH group: 19, Reserved: 0x0
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):Next payload: SA, version: 2.0 Exchange type:
IKE_SA_INIT, flags: RESPONDER MSG-RESPONSE Message id: 0, length: 264
Payload contents:
 SA  Next payload: KE, reserved: 0x0, length: 40
  last proposal: 0x0, reserved: 0x0, length: 36
  Proposal: 1, Protocol id: IKE, SPI size: 0, #trans: 3    last transform: 0x3,
reserved: 0x0: length: 12
    type: 1, reserved: 0x0, id: AES-GCM
    last transform: 0x3, reserved: 0x0: length: 8
    type: 2, reserved: 0x0, id: SHA256
    last transform: 0x0, reserved: 0x0: length: 8
    type: 4, reserved: 0x0, id: DH_GROUP_256_ECP/Group 19
 KE  Next payload: N, reserved: 0x0, length: 72
    DH group: 19, Reserved: 0x0
```

```
AUTH = prf( prf(Shared Secret, "Key Pad for IKEv2"), <InitiatorSignedOctets>)
```

```
AUTH = prf( prf(Cisco123, "Key Pad for IKEv2"), <InitiatorSignedOctets>)
```

```
Type Payload: Authentication (39)
                    Next payload: Security Association (33)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 28
                    Authentication Method: Shared Key Message Integrity Code (2)
                    Authentication Data: 27c5615a79ba2abe1c0077507770d624c3f8e246
```

crypto ikev2 keyring *keyring-name*

```
Router(config)#crypto ikev2 keyring ikev2_keyring
Router(config-ikev2-keyring)#
```

```
Router(config)#crypto ikev2 profile default
Router(config-ikev2-profile)#keyring local ikev2_keyring
```

```
Router(config)#no crypto ikev2 keyring ikev2_keyring
% Cannot remove as keyring is in use.
Router(config)#crypto ikev2 profile default
Router(config-ikev2-profile)#no keyring local ikev2_keyring
Router(config-ikev2-profile)#exit
Router(config)#no crypto ikev2 keyring ikev2_keyring
Router(config)#
```

```
Router(config-ikev2-keyring)#?

IKEv2 Keyring commands:
  exit   Exit from crypto ikev2 keyring sub-mode
  no     Negate a command or set its defaults
  peer   Configure a Peer and associated keys
```

```
Router(config-ikev2-keyring)#peer peer1
Router(config-ikev2-keyring-peer)#?
Crypto IKEv2 Keyring Peer sub-mode commands:
  address          Specify IPv4/IPv6 address of peer
  description      Specify a description of this peer
  exit             Exit from crypto ikev2 keyring peer sub-mode
  hostname         Hostname of peer
  identity         Specify IKE identity to use
  no               Negate values of a command
  pre-shared-key   specify the pre-shared key
```

```
Router(config-ikev2-keyring-peer)#address ?
  A.B.C.D                 IPv4 Address
  X:X:X:X::X/<0-128>      IPv6 address/prefix

Router(config-ikev2-keyring-peer)#address 192.168.1.0 ?
  A.B.C.D  specify mask
  <cr>

Router(config-ikev2-keyring-peer)#address 2001::1/128
```

```
Router(config-ikev2-keyring-peer)#hostname ?
  WORD  Name
```

```
Router(config-ikev2-keyring-peer)#identity ?
  address  IP address
  email    Use email address
  fqdn     Use FQDN
  key-id   proprietary types of identification (ID KEY ID)

Router(config-ikev2-keyring-peer)#identity email ?
  WORD    Specify the name string
  domain  email Domain Name

Router(config-ikev2-keyring-peer)#identity email domain ?
  WORD  Specify the name string

Router(config-ikev2-keyring-peer)#identity fqdn ?
  WORD    Specify the name string
  domain  email Domain Name

Router(config-ikev2-keyring-peer)#identity fqdn domain ?
  WORD  Specify the name string

Router(config-ikev2-keyring-peer)#identity key-id ?
  WORD  ID_KEY_ID <string>

Router(config-ikev2-keyring-peer)#identity address ?
  A.B.C.D     IPv4 Address
  X:X:X:X::X  IPv6 address
```

```
Router(config-ikev2-keyring-peer)#pre-shared-key ?
  0       Specifies an UNENCRYPTED password will follow
  6       Specifies an ENCRYPTED password will follow
  LINE    The UNENCRYPTED (cleartext) user password
  hex     Key entered in hex string
  local   specify signing key
  remote  specify verifying key

Router(config-ikev2-keyring-peer)#pre-shared-key local ?
  0     Specifies an UNENCRYPTED password will follow
  6     Specifies an ENCRYPTED password will follow
  LINE  The UNENCRYPTED (cleartext) user password
  hex   Key entered in hex string

Router(config-ikev2-keyring-peer)#pre-shared-key remote ?
  0     Specifies an UNENCRYPTED password will follow
  6     Specifies an ENCRYPTED password will follow
  LINE  The UNENCRYPTED (cleartext) user password
  hex   Key entered in hex string
```

```
KMI: (Session ID: 3) IPSEC key engine sending message KEY_ENG_REQUEST_SAS to
Crypto IKEv2.
KMI: (Session ID: 3) Crypto IKEv2 received message

IKEv2:% Getting preshared key from profile keyring ikev2_keyring
IKEv2-INTERNAL:% Getting preshared key by address 192.168.1.2
IKEv2:% Matched peer block 'Router2'

IKEv2:(SESSION ID = 3,SA ID = 1):Generating IKE_SA_INIT message
IKEv2:(SESSION ID = 3,SA ID = 1):Sending Packet [To 192.168.1.2:500/From
192.168.1.1:500/VRF i0:f0]
Initiator SPI : 83BA3833DB300CF3 - Responder SPI : 0000000000000000 Message id: 0
IKEv2 IKE_SA_INIT Exchange REQUEST
```

```
KMI: (Session ID: 3) IPSEC key engine sending message KEY_ENG_REQUEST_SAS to
Crypto IKEv2.

KMI: (Session ID: 3) Crypto IKEv2 received message KEY_ENG_REQUEST_SAS from IPSEC
key engine.

IKEv2:% Getting preshared key from profile keyring ikev2_keyring

IKEv2-ERROR:% key not found.

IKEv2-ERROR:Failed to initiate sa
```

```
IKEv2:(SESSION ID = 1069,SA ID = 1):Received Packet [From 192.168.1.1:500/To
192.168.1.2:500/VRF i0:f0]

Initiator SPI : 85A08BDDAE7D63EC - Responder SPI : 2DF24BD4A2F7A66C Message id: 1

IKEv2 IKE_AUTH Exchange REQUEST

IKEv2:(SESSION ID = 1069,SA ID = 1):Searching policy based on peer's identity
'192.168.1.1' of type 'IPv4 address'

IKEv2:found matching IKEv2 profile 'default'

IKEv2-INTERNAL:(1): Choosing IKE profile default

IKEv2:% Getting preshared key from profile keyring ikev2_keyring

IKEv2-INTERNAL:% Getting preshared key by address 192.168.1.1

IKEv2:% Matched peer block 'Router2'
```

```
IKEv2:(SESSION ID = 1090,SA ID = 1):Received Packet [From 192.168.1.1:500/To
192.168.1.2:500/VRF i0:f0]
Initiator SPI : 16B9C6C2858AC956 - Responder SPI : 386B52AB1240F61D Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST

IKEv2:% Getting preshared key from profile keyring ikev2_keyring
IKEv2:(SESSION ID = 1090,SA ID = 1):Get peer's preshared key for 192.168.1.1
IKEv2-INTERNAL:(SESSION ID = 1090,SA ID = 1):Failed to compute authentication data
IKEv2-ERROR:(SESSION ID = 1090,SA ID = 1):: Failed to authenticate the IKE SA
IKEv2:(SESSION ID = 1090,SA ID = 1):Verification of peer's authentication data
FAILED
IKEv2:(SESSION ID = 1090,SA ID = 1):Sending authentication failure notify
```

```
crypto ikev2 keyring ikev2_keyring
 peer abc.com
  address 192.168.1.0 255.255.255.0
  identity fqdn domain abc.com
  identity email domain abc.com
  pre-shared-key local key1
  pre-shared-key remote key2

 peer peer2
  hostname peer2
  address 192.168.2.1 255.255.255.255
  identity address 192.168.2.1
  identity key-id peer2
  identity fqdn peer2.abc.com
  identity email peer2@abc.com
  pre-shared-key key
```

4.4.3. Peer Authorization Database (PAD)

The Peer Authorization Database (PAD) provides the link between the SPD and a security association management protocol such as IKE. It embodies several critical functions:

- identifies the peers or groups of peers that are authorized to communicate with this IPsec entity
- specifies the protocol and method used to authenticate each peer
- provides the authentication data for each peer
- constrains the types and values of IDs that can be asserted by a peer with regard to child SA creation, to ensure that the peer does not assert identities for lookup in the SPD that it is not authorized to represent, when child SAs are created
- peer gateway location info, e.g., IP address(es) or DNS names, MAY be included for peers that are known to be "behind" a security gateway

The PAD provides these functions for an IKE peer when the peer acts as either the initiator or the responder.

crypto ikev2 profile *profile-name*

```
Router(config)#crypto ikev2 profile ikev2_profile
```

IKEv2 profile MUST have:

1. A local and a remote authentication method.
2. A match identity or a match certificate or match any statement.

```
Router(config)#show run | begin ikev2_profile
crypto ikev2 profile ikev2_profile
 ! Profile incomplete (no match identity or match certificate statement)
 ! Profile incomplete (no local and/or remote authentication method specified)
```

```
Router(config)#interface tunnel 100
Router(config-if)tunnel protection ipsec profile ipsec_profile ikev2-profile
ikev2_profile
```

```
Router(config)#crypto ipsec profile ipsec_profile
Router(ipsec-profile)#set ikev2-profile ikev2_profile
Router(config)#interface tunnel 100
Router(config-if)tunnel protection ipsec profile ipsec_profile
```

```
Router(config)#no crypto ikev2 profile ikev2_profile
% Profile ikev2_profile is still in use and cannot be removed
Router(config)#crypto ipsec profile ipsec_profile
Router(ipsec-profile)#no set ikev2-profile ikev2_profile
Router(ipsec-profile)#exit
Router(config)#no crypto ikev2 profile ikev2_profile
Router(config)#
```

```
Router(config-ikev2-profile)#match ?
  address       IP address
  certificate   Peer certificate attributes
  fvrf          fvrf of the profile
  identity      IKE identity
```

```
Router(config)#crypto pki certificate map certificate_map 1
Router(ca-certificate-map)#?
Crypto certificate attribute map commands:
  alt-subject-name           Alternate subject name
  default                    Set a command to its defaults
  exit                       Exit from certificate attribute map mode
  expires-on                 Expiration date/time
  issuer-name                Certificate issuer name
  name                       Any subject name field
  no                         Negate a command or set its defaults
  serial-number              Certificate serial number (hexadecimal)
  subject-name               Certificate subject name
  unstructured-subject-name  Unstructured subject name
  valid-start                Valid after date/time

Router(ca-certificate-map)#subject-name ?
  co  Contains
  eq  Equal
  nc  Does not contain
  ne  Not Equal

Router(ca-certificate-map)#subject-name eq ?
  LINE  Compare string

Router(ca-certificate-map)#subject-name co Router2
Router(ca-certificate-map)#end
Router#
Router(config)#crypto ikev2 profile ikev2_profile
Router(config-ikev2-profile)#match certificate certificate_map
Router(config-ikev2-profile)#
```

```
Router(config-ikev2-profile)#match identity ?
  remote  Remote identity

Router(config-ikev2-profile)#match identity remote ?
  address  IP Address(es)
  any      match any peer identity
  email    Fully qualified email string [Max. 255 char(s)]
  fqdn     Fully qualified domain name string [Max. 255 char(s)]
  key-id   key-id opaque string

Router(config-ikev2-profile)#match identity remote any ?
  <cr>

Router(config-ikev2-profile)#match identity remote address ?
  A.B.C.D              IP address prefix
  X:X:X:X::X/<0-128>   IPv6 address/prefix-length
```

```
Router(config-ikev2-profile)#match identity remote email ?
  WORD    Specify the name string
  domain  match domain instead of exact

Router(config-ikev2-profile)#match identity remote email domain ?
  WORD  Specify the name string

Router(config-ikev2-profile)#match identity remote fqdn ?
  WORD    Specify the name string
  domain  match domain instead of exact

Router(config-ikev2-profile)#match identity remote fqdn domain ?
  WORD  Specify the name string

Router(config-ikev2-profile)#match identity remote key-id ?
  WORD  Specify the key-id string
```

```
Router(config-ikev2-profile)#match identity remote any ?
  <cr>

Router(config-ikev2-profile)#match identity remote any
Router(config-ikev2-profile)#match certificate certificate_map
%Error! please remove 'match identity remote any' statement
Router(config-ikev2-profile)#match identity remote address 192.168.1.1
%Error! please unconfigure 'match identity remote any' in this profile

Router(config-ikev2-profile)#no match identity remote any
Router(config-ikev2-profile)#match identity remote address 192.168.1.1
Router(config-ikev2-profile)#match identity remote any
%Error! please remove 'match certificate' and 'match identity remote' statements
Router(config-ikev2-profile)#
```

```
Router(config-ikev2-profile)#match fvrf ?
  WORD  fvrf name (default: global)
  any   any fvrf

Router(config-ikev2-profile)#match address local ?
  A.B.C.D     Specify an IPv4 address
  X:X:X:X::X  Specify an IPv6 address
  interface   Specify an interface
```

```
Router(config-ikev2-profile)#identity local address ?
  A.B.C.D     IPv4 address
  X:X:X:X::X  IPv6 address

Router(config-ikev2-profile)#identity local dn ?
  <cr>

Router(config-ikev2-profile)#identity local email ?
  WORD  email <user@domain>

Router(config-ikev2-profile)#identity local fqdn ?
  WORD  FQDN <host.domain>

Router(config-ikev2-profile)#identity local key-id ?
  WORD  key-id string
```

```
Router#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         192.168.1.1/500           192.168.1.2/500           none/none
READY
      Encr: AES-CBC, keysize: 256, PRF: SHA512, Hash: SHA512, DH Grp:5, Auth sign:
PSK, Auth verify: PSK
      Life/Active Time: 86400/13 sec
      CE id: 1040, Session-id: 44
      Status Description: Negotiation done
      Local spi: 1935BB7ED97A8119       Remote spi: 4EB9F7349F07B7A4
      Local id: 192.168.1.1
      Remote id: 192.168.1.2
```

```
IKEv2:(SESSION ID = 3077,SA ID = 1):Constructing IDi payload: '192.168.1.1' of
type 'IPv4 address'
IKEv2:(SESSION ID = 3077,SA ID = 1):Searching policy based on peer's identity
'192.168.1.2' of type 'IPv4 address'
```

```
IKEv2:(SESSION ID = 1174,SA ID = 1):Searching policy based on peer's identity
'192.168.1.1' of type 'IPv4 address'
IKEv2:(SESSION ID = 1174,SA ID = 1):Constructing IDr payload: '192.168.1.2' of
type 'IPv4 address'
```

```
IKEv2-ERROR:(SESSION ID = 3077,SA ID = 1):: Detected an unsupported ID type
IKEv2:(SESSION ID = 3077,SA ID = 1):Auth exchange failed
IKEv2:(SESSION ID = 3077,SA ID = 1):Abort exchange
IKEv2:(SESSION ID = 3077,SA ID = 1):Deleting SA
```

```
Type Payload: Authentication (39)
                    Next payload: Security Association (33)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 28
                    Authentication Method: Shared Key Message Integrity Code (2)
                    Authentication Data: 27c5615a79ba2abe1c0077507770d624c3f8e246
```

```
Router(config-ikev2-profile)#authentication local ?
  eap         Extended Authentication Protocol
  ecdsa-sig   ECDSA Signature
  pre-share   Pre-Shared Key
  rsa-sig     Rivest-Shamir-Adleman Signature

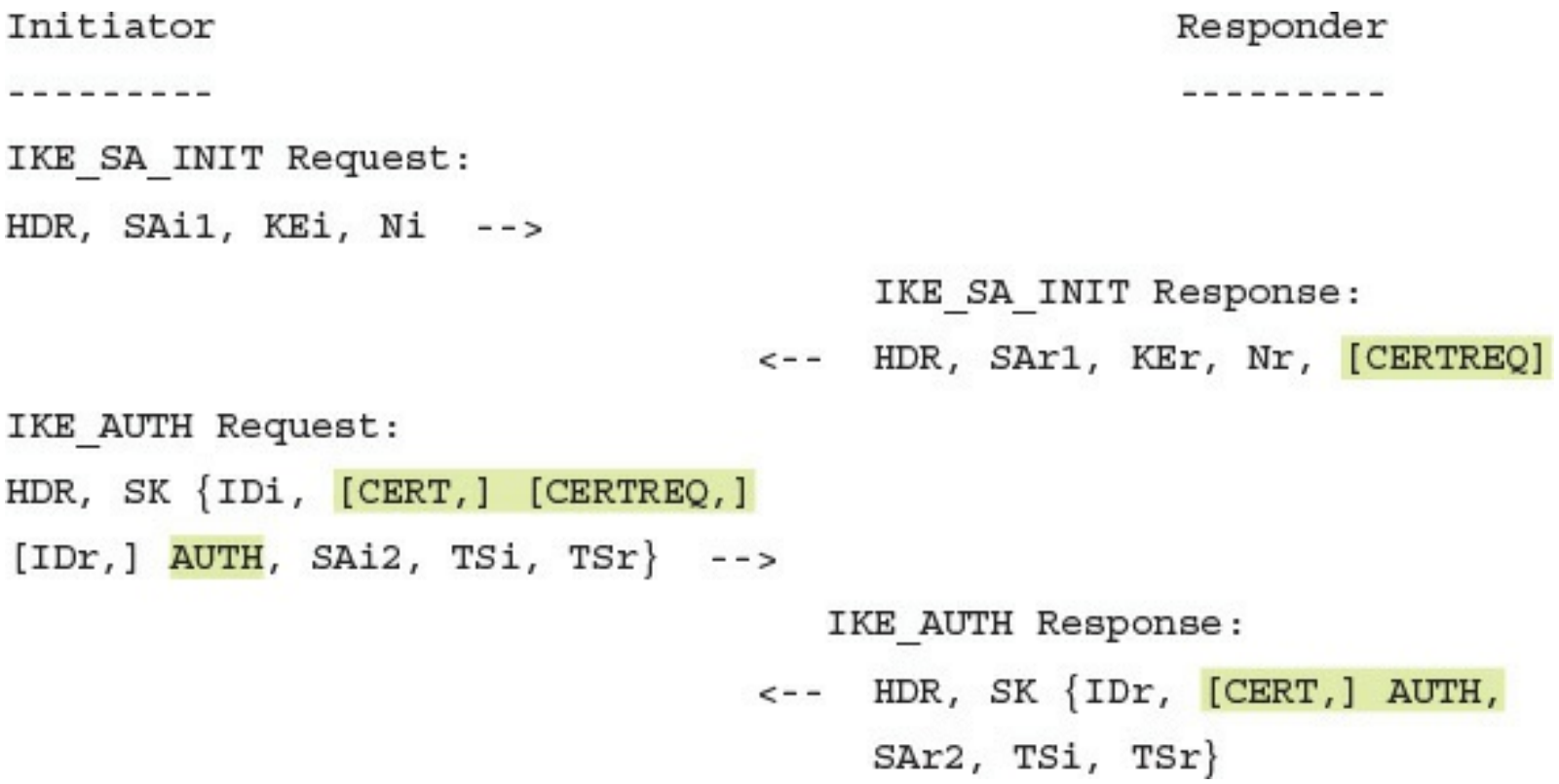
Router(config-ikev2-profile)#authentication remote ?
  anyconnect-eap  AnyConnect EAP
  eap             Extended Authentication Protocol
  ecdsa-sig       ECDSA Signature
  pre-share       Pre-Shared Key
  rsa-sig         Rivest-Shamir-Adleman Signature
```

```
crypto ikev2 profile ikev2_profile
 authentication local pre-share key key1
 authentication remote pre-share key key2
```

```
crypto ikev2 keyring ikev2_keyring
 peer peer1
  address 192.168.2.1 255.255.255.255
  pre-shared-key local key1 remote key2

 peer peer2
  address 192.168.3.0 255.255.255.0
  pre-shared-key local key3 remote key4

crypto ikev2 profile ikev2_profile
 keyring local ikev2_keyring
```


Initiator
Responder
IKE_SA_INIT Request:
HDR, SAi1, KEi, Ni -->
IKE_SA_INIT Response:
<-- HDR, SAr1, KEr, Nr, [CERTREQ]
IKE_AUTH Request:
HDR, SK {IDi, [CERT,] [CERTREQ,]
[IDr,] AUTH, SAi2, TSi, TSr} -->
IKE_AUTH Response:
<-- HDR, SK {IDr, [CERT,] AUTH,
SAr2, TSi, TSr}

```
Router(config-ikev2-profile)#pki trustpoint ca_server1 ?
  sign    Set trustpoint for sign
  verify  Set trustpoint for verify
  <cr>
```

```
crypto pki trustpoint ca_server1
 enrollment url http://192.168.1.4:80
 subject-name cn=Router
 revocation-check crl

crypto ikev2 profile ikev2_profile
 match identity remote any
 authentication local rsa-sig
 authentication remote rsa-sig
 pki trustpoint ca_server1
```

```
Router(config-ikev2-profile)#dpd ?
  <10-3600>  DPD R-U-THERE interval

Router(config-ikev2-profile)#dpd 10 ?
  <2-60>  DPD Retry Interval

Router(config-ikev2-profile)#dpd 10 2 ?
  on-demand  Send DPD queries only as needed
  periodic   Send DPD queries at regular intervals
```

```
crypto ikev2 profile ikev2_profile
   initial-contact force
```

```
Router(config-ikev2-profile)#nat ?
  keepalive  NAT keepalive interval

Router(config-ikev2-profile)#nat keepalive ?
  <5-3600>  NAT keepalive interval in seconds
```

```
Router(config-ikev2-profile)#ivrf ?

  WORD VRF Name
```

```
Router(config-ikev2-profile)#?
  virtual-template  Specify the virtual-template for dynamic interface
                    creation.

Router(config-ikev2-profile)#virtual-template ?
  <1-200>  Virtual-template number


interface Virtual-Template1 type tunnel
 ip unnumbered Ethernet0/0
 tunnel source Ethernet0/0
 tunnel mode ipsec ipv4
 tunnel protection ipsec profile ipsec_profile

crypto ikev2 profile ikev2_profile
virtual-template 1
```

```
Router(config-ikev2-profile)#shutdown ?
  <cr>
```

```
Router#show crypto ikev2 profile

IKEv2 profile: default
 Ref Count: 5
 Match criteria:
  Fvrf: global
  Local address/interface: none
  Identities: any
  Certificate maps: none
 Local identity: none
 Remote identity: none
 Local authentication method: pre-share
 Remote authentication method(s): pre-share
 EAP options: none
 Keyring: ikev2_keyring
 Trustpoint(s): none
 Lifetime: 86400 seconds
 DPD: disabled
 NAT-keepalive: disabled
 Ivrf: none
 Virtual-template: none
 mode auto: none
 AAA AnyConnect EAP authentication mlist: none
 AAA EAP authentication mlist: none
 AAA Accounting: none
 AAA group authorization: none
 AAA user authorization: none
```

```
Router(config)#crypto ikev2 http-url ?
  cert Enable certificate lookup based on HTTP-based URL
```

```
Router(config)#crypto ikev2 ?
  certificate-cache     Cache for storing certs fetched from HTTP URLs

Router(config)#crypto ikev2 certificate-cache ?
  <1-2000>  Maximum number of certificates stored in certificate cache
```

```
Router(config)#crypto ikev2 ?
  cookie-challenge     Set Cookie-challenge watermark

Router(config)#crypto ikev2 cookie-challenge ?
  <0-1000>  Maximum number of in-negotiation SAs that enable Cookie-challenge
```

```
Router(config)#crypto ikev2 ?
  limit                 Limit the number of maximum and negotiating sa

Router(config)#crypto ikev2 limit ?
  max-in-negotation-sa  Limit the maximum number of negotiating SA
  max-sa                Limit the maximum number of SA allowed

Router(config)#crypto ikev2 limit max-sa ?
  <1-9999>  Set the limit

Router(config)#crypto ikev2 limit max-in-negotation-sa ?
  <1-9999>  Set the limit

Router(config)#crypto ikev2 limit max-in-negotation-sa 100 ?
  incoming  Limit the maximum number of incoming negotiating SA
  outgoing  Limit the maximum number of outgoing negotiating SA
  <cr>
```

```
Router#show crypto ikev2 stats
--------------------------------------------------------------------------------
                           Crypto IKEv2 SA Statistics
--------------------------------------------------------------------------------
System Resource Limit:   0        Max IKEv2 SAs: 0        Max in nego(in/out):
100/400
Total incoming IKEv2 SA Count:    0        active:        0        negotiating: 0
Total outgoing IKEv2 SA Count:    1        active:        1        negotiating: 0
Incoming IKEv2 Requests: 0        accepted:      0        rejected:    0
Outgoing IKEv2 Requests: 3        accepted:      3        rejected:    0
Rejected IKEv2 Requests: 0        rsrc low:      0        SA limit:    0
IKEv2 packets dropped at dispatch: 0
Incoming IKEV2 Cookie Challenged Requests: 0
    accepted: 0        rejected: 0        rejected no cookie: 0
Total Deleted sessions of Cert Revoked Peers: 0
```

```
Type Payload: Notify (41)
                    Next payload: Notify (41)
                    0... .... = Critical Bit: Not Critical
                    Payload length: 12
                    Protocol ID: IKE (1)
                    SPI Size: 0
                    Notify Message Type: SET_WINDOW_SIZE (16385)
                    Notification DATA: 00000005
```

IKEv2 window size can be configured globally using the following command:Router(config)**#crypto ikev2 window ?**

```
  <1-200>  Maximum number of overlapping requests with a peer
```

```
Router(config)#crypto ikev2 dpd ?
  <10-3600>  DPD R-U-THERE interval

Router(config)#crypto ikev2 dpd 10 ?
  <2-60>  DPD Retry Interval

Router(config)#crypto ikev2 dpd 10 2 ?
  on-demand  Send DPD queries only as needed
  periodic   Send DPD queries at regular interva
```

```
Router(config)#crypto ikev2 nat keepalive ?
  <5-3600>  Number of seconds between NAT keepalive
```

```
Router(config)#crypto ikev2 diagnose error ?

  <1-1000> Maximum number of exit path entry
```

```
Router#show crypto ikev2 diagnose ?
  error  Shows error trace database

Router#show crypto ikev2 diagnose error ?
  count  Shows ikev2 errors counts - only non-zero
  |      Output modifiers
  <cr>

Router#show crypto ikev2 diagnose error count ?
  detailed  Shows ikev2 errors counts - all
  |         Output modifiers
  <cr>
```

```
Router(config)#crypto pki server CA
Router(cs-server)#?
CA Server configuration commands:
  auto-rollover  Rollover the CA key and certificate
  cdp-url        CRL Distribution Point to be included in the issued certs
  crl            server crl
  database       Certificate Server database config parameters
  default        Set a command to its defaults
  eku            Configure EKU parameters
  exit           Exit from Certificate Server entry mode
  grant          Certificate granting options
  hash           Hash algorithm
  issuer-name    Issuer name
  lifetime       Lifetime parameters
  mode           Mode
  no             Negate a command or set its defaults
  redundancy     sync this server to the standby
  serial-number  Serial Number of Last Certificate Issued
  show           Show this certificate server configuration
  shutdown       Shutdown the Certificate Server
```

```
Router#show crypto pki server

Certificate Server CA:

    Status: disabled, HTTP Server is disabled
```

```
Router(config)#crypto pki server CA
Router(cs-server)#no shut
%Some server settings cannot be changed after CA certificate generation.
% Please enter a passphrase to protect the private key
% or type Return to exit
Password:

Re-enter password:
% Generating 1024 bit RSA keys, keys will be non-exportable...
[OK] (elapsed time was 0 seconds)
```

```
%PKI-6-CS_ENABLED: Certificate server now enabled.

Router#show crypto pki server

Certificate Server CA:

    Status: enabled

    State: enabled

….
```

crypto key generate ec keysize { 256 | 384 } [**exportable**] [**label** *key-label*]

crypto key generate rsa [general-keys | **usage-keys** | **signature** | **encryption]**
[label *key-label* **]** **[exportable]** **[modulus** *modulus-size* **]** **[storage** *devicename* **:]**
[redundancy] **[on** *devicename* **:]**

crypto key import ec *key-label* [**exportable**] { **terminal** | **url** *url* } *passphrase*

crypto key import rsa key-label pem [usage-keys | signature | encryption | general-purpose] { storage | terminal *[passphrase]* **| url url } [exportable] [on** *devicename* **:]**

crypto key zeroize ec *[key-pair-label]*

crypto key zeroize rsa *[key-pair-label]*

```
Router(config)#hostname CA
CA(config)#ip domain-name cisco.com
CA(config)#crypto key generate rsa general-keys modulus 2048 label CA
The name for the keys will be: CA

% The key modulus size is 2048 bits
% Generating 2048 bit RSA keys, keys will be non-exportable...
[OK] (elapsed time was 1 seconds)

CA(config)#crypto pki trustpoint CA
CA(ca-trustpoint)#rsakeypair CA
CA(ca-trustpoint)#exit
CA(config)#crypto pki server CA
CA(cs-server)#grant auto
%PKI-6-CS_GRANT_AUTO: All enrollment requests will be automatically granted.
CA(cs-server)#no shutdown
%Some server settings cannot be changed after CA certificate generation.
% Please enter a passphrase to protect the private key
% or type Return to exit
Password:

Re-enter password:

% Certificate Server enabled.
CA(cs-server)#
%PKI-6-CS_ENABLED: Certificate server now enabled.

CA#show crypto pki server
Certificate Server CA:
    Status: enabled
    State: enabled
    Server's configuration is locked  (enter "shut" to unlock it)
    Issuer name: CN=CA
    CA cert fingerprint: 1CB1376E 48B40FFC 8C68A493 D744A83A
    Granting mode is: auto
```

```
Router(config)#crypto pki trustpoint PKI
Router(ca-trustpoint)#enrollment url http://192.168.1.3
Router(ca-trustpoint)#exit
Router(config)#crypto pki authenticate PKI
Certificate has the following attributes:
       Fingerprint MD5: 1CB1376E 48B40FFC 8C68A493 D744A83A
      Fingerprint SHA1: 805741B2 A094DFC3 CAFF1D7D 819EA99F B9C7879E

% Do you accept this certificate? [yes/no]: yes
Trustpoint CA certificate accepted.
Router(config)#crypto pki enroll PKI
%
% Start certificate enrollment ..
% Create a challenge password. You will need to verbally provide this
   password to the CA Administrator in order to revoke your certificate.
   For security reasons your password will not be saved in the configuration.
   Please make a note of it.

Password:
Re-enter password:

% The subject name in the certificate will include: CA.cisco.com
% Include the router serial number in the subject name? [yes/no]:
no
% Include an IP address in the subject name? [no]:
Request certificate from CA? [yes/no]: yes
% Certificate request sent to Certificate Authority
% The 'show crypto pki certificate verbose PKI' command will show the fingerprint.

Router(config)#
CRYPTO_PKI:  Certificate Request Fingerprint MD5: 552781FF 2584FBB1 9E014F06
51BCB61D
CRYPTO_PKI:  Certificate Request Fingerprint SHA1: 70190AD2 B255D1A9 03E942F8
A5F97C1F 5078E579
Router(config)#
%PKI-6-CERTRET: Certificate received from Certificate Authority
```

```
crypto ikev2 profile ikev2_profile
 match identity remote any
 authentication local pre-share key key1
 authentication remote pre-share key key1

crypto ipsec transform-set transform_set esp-aes esp-sha-hmac
 mode transport

crypto ipsec profile ipsec_profile
 set transform-set transform_set
 set ikev2-profile default

interface Tunnel0
 ip address 192.168.0.1 255.255.255.0
 tunnel source 172.16.1.1
 tunnel destination 172.16.2.1
 tunnel protection ipsec profile ipsec_profile
```

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key key1
 authentication remote pre-share key key1

interface Tunnel0
 ip address 192.168.0.1 255.255.255.0
 tunnel source 172.16.1.1
 tunnel destination 172.16.2.1
 tunnel protection ipsec profile ipsec_profile
```

```
Router#show crypto ikev2 proposal default
 IKEv2 proposal: default
      Encryption : AES-CBC-256 AES-CBC-192 AES-CBC-128
      Integrity  : SHA512 SHA384 SHA256 SHA96 MD596
      PRF        : SHA512 SHA384 SHA256 SHA1 MD5
      DH Group   : DH_GROUP_1536_MODP/Group 5 DH_GROUP_1024_MODP/Group 2

Router#show crypto ikev2 policy default
 IKEv2 policy : default
       Match fvrf : any
       Match address local : any
       Proposal    : default

Router#show crypto ipsec profile default
IPSEC profile default
        IKEv2 Profile: default
        Security association lifetime: 4608000 kilobytes/3600 seconds
        Responder-Only (Y/N): N
        PFS (Y/N): N
        Mixed-mode : Disabled
        Transform sets={
                default:  { esp-aes esp-sha-hmac  } ,
        }
```

```
Router#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         10.10.10.1/500        10.10.50.5/500        OUTSIDE/none         READY
      Encr: AES-CBC, keysize: 128, PRF: SHA256, Hash: SHA256, DH Grp:14, Auth
sign: RSA, Auth verify: RSA
      Life/Active Time: 120/38 sec
      CE id: 1074, Session-id: 1171
      Status Description: Negotiation done
      Local spi: F78AC1F7C7C9F558       Remote spi: 5043A7F7F7E1A622
      Local id: hostname=R1.cisco.com
      Remote id: hostname=R5.cisco.com,cn=r5,o=cisco,ou=service1
      Local req msg id:  0              Remote req msg id:  3
      Local next msg id: 0              Remote next msg id: 3
      Local req queued:  0              Remote req queued:  3
      Local window:      5              Remote window:      5
      DPD configured for 0 seconds, retry 0
      IETF Std Fragmentation  enabled.
      IETF Std Fragmentation MTU in use: 972 bytes.…
```

```
Router#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         10.10.10.1/500        10.10.50.5/500        OUTSIDE/none         READY
      Encr: AES-CBC, keysize: 128, PRF: SHA256, Hash: SHA256, DH Grp:14, Auth
sign: RSA, Auth verify: RSA
      Life/Active Time: 120/38 sec
      CE id: 1074, Session-id: 1171
      Status Description: Negotiation done
      Local spi: F78AC1F7C7C9F558       Remote spi: 5043A7F7F7E1A622
      Local id: hostname=R1.cisco.com
      Remote id: hostname=R5.cisco.com,cn=r5,o=cisco,ou=service1
      Local req msg id:  0              Remote req msg id:  3
      Local next msg id: 0              Remote next msg id: 3
      Local req queued:  0              Remote req queued:  3
      Local window:      5              Remote window:      5
      DPD configured for 0 seconds, retry 0
Fragmentation not configured.
      Extended Authentication not configured.
      NAT-T is not detected
      Cisco Trust Security SGT is enabled
```

IKEv2:Sending Delete Reason Notify with error code: **IKEV2_DELETE_CERT_REVOKED** severity: INFORMATIONAL

IKEv2:Adjusting lifetime due to Cert Validity date

```
IKEv2:(SESSION ID = 24,SA ID = 1):Queuing IKE SA delete request reason:
certificate lifetime expired
```

session-lifetime *<120- 25920000>*

```
Router#show crypto session
Crypto session current status

Interface: Virtual-Access1
Profile: default
Session status: UP-ACTIVE
Peer: 10.10.50.5 port 500
  Session ID: 3
  Session Lifetime: 00:05:22
  IKEv2 SA: local 10.10.10.1/500 remote 10.10.50.5/500 Active
  IPSEC FLOW: permit 47 host 10.10.10.1 host 10.10.50.5
        Active SAs: 2, origin: crypto-map

Interface: Virtual-Access2
Profile: default
Session status: UP-ACTIVE
Peer: 10.10.60.6 port 500
  Session ID: 6
  IKEv2 SA: local 10.10.10.1/500 remote 10.10.60.6/500 Active
  IPSEC FLOW: permit 47 host 10.10.10.1 host 10.10.60.6
        Active SAs: 2, origin: crypto-map
```

```
IKEv2:Adjusting lifetime due to Session expiry time

IKEv2:(SESSION ID = 7,SA ID = 1):IKEV2 SA created; inserting SA into database. SA
lifetime timer (600 sec) started

%IKEV2-5-SA_UP: SA UP
```

```
(SESSION ID = 11,SA ID = 1):Queuing IKE SA delete request reason: session lifetime
expired
```

```
crypto ikev2 keyring local_keyring
 peer 2001:DB8::2
  address 2001:DB8::2/128
  pre-shared-key local bartlett
  pre-shared-key remote inamdar
```

```
crypto ikev2 profile default
 match identity remote address 2001:DB8::2/128
 identity local address 2001:DB8::1
 authentication remote pre-share
 authentication local pre-share
 keyring local local_keyring
```

```
interface Loopback0
 ip address 192.168.10.1 255.255.255.0
```

```
interface Tunnel0
 ip address 172.16.1.1 255.255.255.252
 tunnel source Ethernet0/0
 tunnel mode gre ipv6
 tunnel destination 2001:DB8::2
 tunnel protection ipsec profile default
```

```
interface Ethernet0/0
 no ip address
 ipv6 address 2001:DB8::1/64
```

```
router eigrp 1
 network 172.16.1.0 0.0.0.3
 network 192.168.10.0
```

```
interface Loopback0
 ip address 192.168.20.1 255.255.255.0

interface Tunnel0
 ip address 172.16.1.2 255.255.255.252
 tunnel source Ethernet0/0
 tunnel mode gre ipv6
 tunnel destination 2001:DB8::1
 tunnel protection ipsec profile default

interface Ethernet0/0
 no ip address
 ipv6 address 2001:DB8::2/64

router eigrp 1
 network 172.16.1.0 0.0.0.3
 network 192.168.20.0
```

```
Router1#show ip route 192.168.20.0
Routing entry for 192.168.20.0/24
  Known via "eigrp 1", distance 90, metric 27008000, type internal
  Redistributing via eigrp 1
  Last update from 172.16.1.2 on Tunnel0, 00:12:04 ago
  Routing Descriptor Blocks:
  * 172.16.1.2, from 172.16.1.2, 00:12:04 ago, via Tunnel0
      Route metric is 27008000, traffic share count is 1
      Total delay is 55000 microseconds, minimum bandwidth is 100 Kbit
      Reliability 255/255, minimum MTU 1418 bytes
      Loading 1/255, Hops 1
```

```
Router1#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

 IPv6 Crypto IKEv2  SA

Tunnel-id    fvrf/ivrf              Status
1          none/none              READY
Local  2001:DB8::1/500
Remote  2001:DB8::2/500
      Encr: AES-CBC, keysize: 256, PRF: SHA512, Hash: SHA512, DH Grp:5, Auth sign:
PSK, Auth verify: PSK
      Life/Active Time: 86400/10523 sec
      CE id: 1002, Session-id: 2
      Status Description: Negotiation done
      Local spi: 261B9BD2F208A02A       Remote spi: 0B28D2A21FC6304D
      Local id: 2001:DB8::1
      Remote id: 2001:DB8::2
      Local req msg id:  4              Remote req msg id:  4
      Local next msg id: 4              Remote next msg id: 4
      Local req queued:  4              Remote req queued:  4
      Local window:      5              Remote window:      5
…
```

```
Router1#ping 192.168.20.1 source 192.168.10.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.20.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.10.1

Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/5 ms
```

```
Router1#show crypto ipsec sa

interface: Tunnel0
    Crypto-map tag: Tunnel0-head-0, local addr 2001:DB8::1

   protected vrf: (none)
   local  ident (addr/mask/prot/port): (2001:DB8::1/128/47/0)
   remote ident (addr/mask/prot/port): (2001:DB8::2/128/47/0)
   current_peer 2001:DB8::2 port 500
     PERMIT, flags={origin_is_acl,}
    #pkts encaps: 523, #pkts encrypt: 523, #pkts digest: 523
    #pkts decaps: 523, #pkts decrypt: 523, #pkts verify: 523
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0
    #pkts not decompressed: 0, #pkts decompress failed: 0
    #send errors 0, #recv errors 0

     local crypto endpt.: 2001:DB8::1,
     remote crypto endpt.: 2001:DB8::2
     plaintext mtu 1462, path mtu 1500, ipv6 mtu 1500, ipv6 mtu idb Ethernet0/0
     current outbound spi: 0x5FC3C94A(1606666570)
     PFS (Y/N): N, DH group: none

     inbound esp sas:
      spi: 0xB8435B94(3091422100)
```

```
        transform: esp-aes esp-sha-hmac ,
        in use settings ={Transport, }
        conn id: 10, flow_id: SW:10, sibling_flags 80000001, crypto-map:
Tunnel0-head-0
        sa timing: remaining key lifetime (k/sec): (4315844/2543)
        IV size: 16 bytes
        replay detection support: Y
        Status: ACTIVE(ACTIVE)

     inbound ah sas:

     inbound pcp sas:

     outbound esp sas:
      spi: 0x5FC3C94A(1606666570)
        transform: esp-aes esp-sha-hmac ,
        in use settings ={Transport, }
        conn id: 9, flow_id: SW:9, sibling_flags 80000001, crypto-map:
Tunnel0-head-0
        sa timing: remaining key lifetime (k/sec): (4315844/2543)
        IV size: 16 bytes
        replay detection support: Y
        Status: ACTIVE(ACTIVE)
```

```
crypto pki trustpoint ecdsa_tp
 enrollment terminal

crypto pki certificate chain ecdsa_tp
 certificate 6156E3D5000000000009
  308203BF 30820365 A0030201 02020A61 56E3D500 00000000 09300A06 082A8648
…
  68656c6c 6f736861 627333E4FDDC 642DA416 F57D4962 C5DF6545 FEC931FA F84BAF40
  A9829E
      quit
 certificate ca 780887F0CDD97E9E49DB893FA5D74238
  30820206 308201AB A0030201 02021078 0887F0CD D97E9E49 DB893FA5 D7423830
  0A06082A 8648CE3D 04030230 4F311330 11060A09 92268993 F22C6401 19160363
…
  816AA443 9191FBAC 731C
      quit
```

```
crypto pki certificate map certmap 10
 subject-name co cisco.com
```

```
no crypto ikev2 proposal default

crypto ikev2 proposal nge
 encryption aes-gcm-256
 prf sha512
 group 21
```

```
crypto ikev2 profile nge
 match certificate certmap
 identity local dn
 authentication remote ecdsa-sig
 authentication local ecdsa-sig
 pki trustpoint ecdsa_tp
 dpd 10 2 on-demand
```

```
crypto ipsec transform-set nge-transform esp-gcm 256
 mode transport
```

```
no crypto ipsec profile default


crypto ipsec profile nge-profile
 set transform-set nge-transform
 set pfs group21
 set ikev2-profile nge
```

```
interface Loopback0
 ip address 192.168.10.1 255.255.255.0
```

```
interface Tunnel0
 ip address 172.16.1.1 255.255.255.252
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.2
 tunnel protection ipsec profile nge-profile
```

```
interface Ethernet0/0
 ip address 10.10.10.1 255.255.255.0
```

```
ip route 192.168.20.0 255.255.255.0 172.16.1.2
```

```
interface Tunnel0
 ip address 172.16.1.2 255.255.255.252
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.1
 tunnel protection ipsec profile nge-profile

interface Ethernet0/0
 ip address 10.10.10.2 255.255.255.0
```

```
Router1#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         10.10.10.1/500        10.10.10.2/500        none/none            READY
      Encr: AES-GCM, keysize: 256, PRF: SHA512, Hash: None, DH Grp:21, Auth sign:
ECDSA, Auth verify: ECDSA
      Life/Active Time: 86400/6 sec
      CE id: 1030, Session-id: 13
      Status Description: Negotiation done
      Local spi: 313404E23B3A5707       Remote spi: 13FE5BCC09FFAAAB
      Local id: hostname=Router1.cisco.com
      Remote id: hostname=Router2.cisco.com
      Local req msg id:  2              Remote req msg id:  0
      Local next msg id: 2              Remote next msg id: 0
      Local req queued:  2              Remote req queued:  0
      Local window:      5              Remote window:      5
      DPD configured for 10 seconds, retry 2
      Fragmentation not configured.
      Extended Authentication not configured.
      NAT-T is not detected
      Cisco Trust Security SGT is disabled
      Initiator of SA : Yes

 IPv6 Crypto IKEv2  SA
```

```
Router1#show crypto sockets

Number of Crypto Socket connections 1

   Tu0 Peers (local/remote): 10.10.10.1/10.10.10.2
       Local Ident  (addr/mask/port/prot): (10.10.10.1/255.255.255.255/0/47)
       Remote Ident (addr/mask/port/prot): (10.10.10.2/255.255.255.255/0/47)
       IPSec Profile: "nge-profile"
       Socket State: Open
       Client: "TUNNEL SEC" (Client State: Active)
Crypto Sockets in Listen state:
Client: "TUNNEL SEC" Profile: "nge-profile" Map-name: "Tunnel0-head-0"
```

```
Router1#show ip route 192.168.20.0 255.255.255.0
Routing entry for 192.168.20.0/24
  Known via "static", distance 1, metric 0
  Routing Descriptor Blocks:
  * 172.16.1.2
      Route metric is 0, traffic share count is 1
Router1#show ip route 172.16.1.2
Routing entry for 172.16.1.0/30
  Known via "connected", distance 0, metric 0 (connected, via
  Routing Descriptor Blocks:
  * directly connected, via Tunnel0
      Route metric is 0, traffic share count is 1
```

```
Router1#ping 192.168.20.1 source 192.168.10.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.20.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.10.1

Success rate is 100 percent (5/5), round-trip min/avg/max = 3/4/6 ms

Router1#show crypto ipsec sa

interface: Tunnel0
    Crypto-map tag: Tunnel0-head-0, local addr 10.10.10.1

   protected vrf: (none)
   local  ident (addr/mask/prot/port): (10.10.10.1/255.255.255.255/47/0)
   remote ident (addr/mask/prot/port): (10.10.10.2/255.255.255.255/47/0)
   current_peer 10.10.10.2 port 500
     PERMIT, flags={origin_is_acl,}
    #pkts encaps: 10, #pkts encrypt: 10, #pkts digest: 10
    #pkts decaps: 10, #pkts decrypt: 10, #pkts verify: 10
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0
```

```
    #pkts not decompressed: 0, #pkts decompress failed: 0
    #send errors 0, #recv errors 0

     local crypto endpt.: 10.10.10.1, remote crypto endpt.: 10.10.10.2
     plaintext mtu 1466, path mtu 1500, ip mtu 1500, ip mtu idb Ethernet0/0
     current outbound spi: 0x3FD4A2AF(1070899887)
     PFS (Y/N): Y, DH group: group21

     inbound esp sas:
      spi: 0x349334C6(882062534)
        transform: esp-gcm 256 ,
        in use settings ={Transport, }
        conn id: 6, flow_id: SW:6, sibling_flags 80000000, crypto-map:
Tunnel0-head-0
        sa timing: remaining key lifetime (k/sec): (4207250/3566)
        IV size: 8 bytes
        replay detection support: Y
        Status: ACTIVE(ACTIVE)

     inbound ah sas:

     inbound pcp sas:

     outbound esp sas:
      spi: 0x3FD4A2AF(1070899887)
        transform: esp-gcm 256 ,
        in use settings ={Transport, }
```

```
        conn id: 5, flow_id: SW:5, sibling_flags 80000000, crypto-map:
Tunnel0-head-0
        sa timing: remaining key lifetime (k/sec): (4207250/3566)
        IV size: 8 bytes
        replay detection support: Y
        Status: ACTIVE(ACTIVE)

    outbound ah sas:

    outbound pcp sas:
```

```
openssl x509 -outform der -in 3.crt -out 3.der
```

```
crypto pki trustpoint CA
 enrollment url http://10.10.10.1:80
 revocation-check crl
 match certificate local override sia 1 http://192.168.1.100/3.der
```

```
crypto pki certificate map certmap 10
 subject-name eq router1.cisco.com
```

```
crypto pki certificate map local 10
 subject-name co router2
```

```
crypto ikev2 profile default
 match certificate certmap
 identity local dn
 authentication remote rsa-sig
 authentication local rsa-sig
 pki trustpoint CA
```

```
interface Tunnel0
 ip address 172.16.1.2 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.1
 tunnel protection ipsec profile default
```

```
interface Ethernet0/0
 ip address 10.10.10.2 255.255.255.0
```

```
crypto pki trustpoint CA
 match certificate local override sia 1 http://192.168.1.100/3.der
 match certificate local override sia 2 http://192.168.1.100/subca1.der
 match certificate local override sia 3 http://192.168.1.100/subca2.der
 match certificate local override sia 4 http://192.168.1.100/root.der
```

```
crypto pki trustpoint CA
 enrollment url http://10.10.10.1:80
 revocation-check crl
```

```
crypto pki certificate map certmap 10
 subject-name eq router2.cisco.com
```

```
crypto ikev2 profile default
 match certificate certmap
 identity local dn
 authentication remote rsa-sig
 authentication local rsa-sig
 pki trustpoint CA
```

```
interface Tunnel1
 ip address 172.16.1.1 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.2
 tunnel protection ipsec profile default
```

```
interface Ethernet0/0
 ip address 10.10.10.1 255.255.255.0
```

```
interface Ethernet0/1
 ip address 192.168.1.1 255.255.255.0
```

```
IKEv2:(SESSION ID = 4,SA ID = 1):Received Packet [From 10.10.10.2:500/To
10.10.10.1:500/VRF i0:f0]
Initiator SPI : 52D538043A8E330C - Responder SPI : 5CE063D07E8745EA Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST
IKEv2-PAK:(SESSION ID = 4,SA ID = 1):Next payload: ENCR, version: 2.0 Exchange
type: IKE_AUTH, flags: INITIATOR Message id: 1, length: 816
Payload contents:
 VID  Next payload: IDi, reserved: 0x0, length: 20
 IDi  Next payload: CERT, reserved: 0x0, length: 44
    Id type: DER ASN1 DN, Reserved: 0x0 0x0
 CERT  Next payload: CERTREQ, reserved: 0x0, length: 52
    Cert encoding Hash and URL of PKIX
 CERTREQ  Next payload: NOTIFY, reserved: 0x0, length: 25
    Cert encoding Hash and URL of PKIX
 NOTIFY(HTTP_CERT_LOOKUP_SUPPORTED)  Next payload: AUTH, reserved: 0x0, length: 8
    Security protocol id: Unknown - 0, spi size: 0, type:
HTTP_CERT_LOOKUP_SUPPORTED
 AUTH  Next payload: CFG, reserved: 0x0, length: 136
    Auth method RSA, reserved: 0x0, reserved 0x0
 CFG  Next payload: SA, reserved: 0x0, length: 304
    cfg type: CFG_REQUEST, reserved: 0x0, reserved: 0x0
```

```
TCP: sending SYN, seq 2191097267, ack 0
TCP0: Connection to 192.168.1.100:80, advertising MSS 1460
TCP0: state was CLOSED -> SYNSENT [42603 -> 192.168.1.100(80)]
TCP0: state was SYNSENT -> ESTAB [42603 -> 192.168.1.100(80)]
TCP: tcb 32417230 connection to 192.168.1.100:80, peer MSS 1460, MSS is 1460
```

```
Router1#show crypto ikev2 stats ext-service
```

AAA OPERATION	PASSED	FAILED
RECEIVING PSKEY	0	0
AUTHENTICATION USING EAP	0	0
START ACCOUNTING	0	0
STOP ACCOUNTING	0	0
AUTHORIZATION	0	0

IPSEC OPERATION	PASSED	FAILED
IPSEC POLICY VERIFICATION	3	0
SA CREATION	3	0
SA DELETION	3	0

CRYPTO ENGINE OPERATION	PASSED	FAILED
DH PUBKEY GENERATED	34	0

DH SHARED SECKEY GENERATED	29	0
SIGNATURE SIGN	28	0
SIGNATURE VERIFY	3	0

PKI OPERATION	PASSED	FAILED
VERIFY CERTIFICATE	3	0
FETCHING CERTIFICATE USING HTTP	1	0
FETCHING PEER CERTIFICATE USING HTTP	1	0
GET ISSUERS	31	0
GET CERTIFICATES FROM ISSUERS	28	0
GET DN FROM CERT	3	0

GKM OPERATION	PASSED	FAILED
GET_POLICY	0	0
SET_POLICY	0	0

```
Router1#show crypto ikev2 certificate-cache
No of entries in ikev2 certificate-cache = 1

Certificate entry:
Certificate
  Status: Available
  Certificate Serial Number (hex): 03
  Certificate Usage: General Purpose
  Issuer:
    cn=CA.cisco.com
  Subject:
    Name: Router2.cisco.com
    hostname=Router2.cisco.com
  Validity Date:
    start date: 10:44:26 UTC Feb 8 2016
    end   date: 10:44:26 UTC Feb 7 2017
  Associated Trustpoints:
```

```
Router1#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         10.10.10.1/500        10.10.10.2/500        none/none            READY
      Encr: AES-CBC, keysize: 256, PRF: SHA512, Hash: SHA512, DH Grp:5, Auth sign:
RSA, Auth verify: RSA
      Life/Active Time: 86400/509 sec
      CE id: 1034, Session-id: 7
      Status Description: Negotiation done
      Local spi: 5CE063D07E8745EA       Remote spi: 52D538043A8E330C
      Local id: hostname=Router1.cisco.com
      Remote id: hostname=Router2.cisco.com
      Local req msg id:  0              Remote req msg id:  2
      Local next msg id: 0              Remote next msg id: 2
      Local req queued:  0              Remote req queued:  2
      Local window:      5              Remote window:      5
      DPD configured for 0 seconds, retry 0
      Fragmentation not  configured.
      Extended Authentication not configured.
      NAT-T is not detected
      Cisco Trust Security SGT is disabled
      Initiator of SA : No
```

```
Router#show processes cpu history

  100
   90            *****
   80            *****                                *****
   70            *****                                *****
   60            *****                                *****
   50            **********                           **********
   40            **********                           **********
   30            **********                           **********
   20            **********                           **********
   10            **********                 ***************
     0....5....1....1....2....2....3....3....4....4....5....5....6
                0    5    0    5    0    5    0    5    0    5    0
                CPU% per second (last 60 seconds)
```

```
Router(config)#crypto ikev2 limit max-in-negotation-sa 1000
```

```
Router(config)#crypto ikev2 cookie-challenge 0
```

```
Router#show processes cpu history


  100       ************************************************************
   90       ************************************************************
   80       ************************************************************
   70       ************************************************************
   60       ************************************************************
   50       ************************************************************
   40       ************************************************************
   30       ************************************************************
   20       ************************************************************
   10       ************************************************************
     0....5....1....1....2....2....3....3....4....4....5....5....6
                0    5    0    5    0    5    0    5    0    5    0
                     CPU% per second (last 60 seconds)
```

```
interface tunnel1
 vrf forwarding ivrf_1
 ip address 192.168.1.1 255.255.255.0
 ipv6 address 2001::1:1/120
 tunnel source Ethernet0/0
 tunnel destination 192.168.2.1
 tunnel vrf fvrf_1
 tunnel protection ipsec profile default
end

Router#show interfaces tunnel 1 | include Tunnel protocol/transport
  Tunnel protocol/transport GRE/IP
```

```
Router(config)#interface tunnel 1
Router(config-if)#tunnel mode gre ?
  ip            over IP
  ipv6          over IPv6

Router (config-if)#tunnel mode ipsec ?
  ipv4   over IPv4
  ipv6   over IPv6
```

```
crypto ikev2 profile ikev2_profile
 virtual-template 1

interface virtual-template1 type tunnel
 ip unnumbered Ethernet0/0
 ipv6 unnumbered Ethernet0/0
 tunnel protection ipsec profile default

Router#show interfaces virtual-template 1 | include Tunnel protocol
  Tunnel protocol/transport GRE/IP
```

```
Router(config)#interface virtual-template 1 type tunnel
Router(config-if)#tunnel mode gre ?
  ip          over IP
  ipv6        over IPv6

Router (config-if)#tunnel mode ipsec ?
  ipv4  over IPv4
  ipv6  over IPv6
```

```
Router(config)#crypto ikev2 profile ikev2_profile
Router(config-ikev2-profile)#virtual-template 1 ?
  mode  Enabling Tunnel Auto Mode
  <cr>

Router(config-ikev2-profile)#virtual-template 1 mode ?
  auto  Auto - Enable Auto feature on the ike profile

Router(config-ikev2-profile)#virtual-template 1 mode auto
Warning! Mode auto takes precedence over the mode configured on the
Virtual-Template
```

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share
 authentication remote pre-share
 keyring local ikev2_keyring
 virtual-template 1 mode auto

interface Virtual-Template1 type tunnel
 ip unnumbered Ethernet0/0
 tunnel protection ipsec profile default
```

```
IKEv2:(SESSION ID = 17,SA ID = 1):Processing IKE_AUTH message
IKEv2:% DVTI create request sent for profile default with PSH index 1.
 %LINEPROTO-5-UPDOWN: Line protocol on Interface Virtual-Access1, changed state to
down
VT[Vi1]:Clone Vaccess from Virtual-Template1 (96 bytes)
VT[Vi1]:tunnel mode ipsec ipv6
VT[Vi1]:tunnel protection ipsec profile default
VT[Vi1]:end
IKEv2:% DVTI Vi1 created for profile default with PSH index 1.
```

```
Router#show crypto session
Crypto session current status
Interface: Virtual-Access1
Profile: default
Session status: UP-ACTIVE
Peer: 172.16.1.1 port 500
  Session ID: 15
  IKEv2 SA: local 172.16.2.1/500 remote 172.16.1.1/500 Active
  IPSEC FLOW: permit 47 host 172.16.2.1 host 172.16.1.1
        Active SAs: 2, origin: crypto-map
```

```
Router#show derived-config interface virtual-access 1
Building configuration...
Derived configuration : 218 bytes

interface Virtual-Access1
 ip unnumbered Ethernet0/0
 tunnel source 172.16.2.1
 tunnel destination 172.16.1.1
 tunnel protection ipsec profile default
 no tunnel protection ipsec initiate
end
```

```
aaa authentication login authen_list_radius group radius_group
aaa authorization network author_list_local local
aaa authorization network author_list_radius group radius_group
aaa accounting network acc_list_radius group radius_group
```

```
aaa group server radius radius_group
 server name radius_server
```

```
radius server radius_server
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_key
```

```
crypto ikev2 authorization policy author_policy
 route set interface
 route accept any
 pool ip_address_pool
 aaa attribute list attr_list
```

```
aaa attribute list attr-list1
attribute type interface-config "ip mtu 1100"
attribute type interface-config "tunnel key 10"
```

```
crypto ikev2 profile ikev2_profile
 aaa authentication eap authen_list_radius
 aaa authorization user cert list author_list_radius user1
 aaa authorization group cert list author_list_local author_policy
 aaa accounting cert acc_list_radius
```

crypto ikev2 name-mangler *name_mangler*

```
Router(config-ikev2-name-mangler)#?
IKEv2 name mangler commands:
  dn      Derive name from DN identity
  eap     Derive name from EAP identity
  email   Derive name from EMAIL identity
  fqdn    Derive name from FQDN identity

Router#show run | begin name-mangler
crypto ikev2 name-mangler name_mangler
 fqdn hostname
 email domain
 dn common-name
 eap all
```

```
crypto ikev2 profile ikev2_profile
 keyring aaa aaa_method_list name-mangler name_mangler
 aaa authorization group psk list aaa_list name-mangler name_mangler
 aaa authorization user psk list aaa_list name-mangler name_mangler
```

```
Router(config)#no cryto ikev2 name-mangler name_mangler
% Cannot remove as name mangler is in use.
```

```
Router(config)#crypto ikev2 authorization policy ikev2_author_policy
Router(config-ikev2-author-policy)#?
IKEv2 authorization policy commands:
  aaa                           Specify aaa attribute list
  backup-gateway                Specify backup gateway
  banner                        Specify mode config banner
  configuration                 Push configuration to the client
  def-domain                    Set default domain name to send to client
  dhcp                          Specify DHCP server for config address
                                assignment
  dns                           Specify DNS Addresses
  exit                          Exit from crypto ikev2 author policy sub mode
  include-local-lan             Enable Local LAN Access with no split tunnel
  ipsec                         Specify ipsec parameters
  ipv6                          Specify the ipv6 attributes
  mobileopt-serv                Specify Mobile Optimization Server Addr and
                                Port
  netmask                       Specify netmask of the config address
  no                            Negate a command or set its defaults
  pfs                           The client should propose PFS
  pool                          Specify local address pool
  route                         Specify route parameters
  session-lifetime              Specify maximum session lifetime
  smartcard-removal-disconnect  Enables smartcard-removal-disconnect
  split-dns                     Split DNS names
  wins                          Specify WINS Addresses
```

```
Router(config-ikev2-author-policy)#aaa attribute list ?
  WORD  AAA attribute list name
```

```
aaa attribute list attr-list1
 attribute type interface-config "ip mtu 1100"
 attribute type interface-config "tunnel key 10"
```

```
aaa new-model

aaa authorization network aaa_list1 local
```

```
crypto ikev2 profile default
 match identity remote any
 identity local dn
 authentication local pre-share
 authentication remote pre-share
 aaa authorization group psk list aaa_list1 ikev2_author_policy
 aaa authorization user psk list aaa_list1 ikev2_author_policy
```

```
Router#show crypto ikev2 authorization policy
 IKEv2 Authorization Policy : default
  route set interface
  route accept any tag : 1 distance : 1
 IKEv2 Authorization Policy : ikev2_author_policy
  IPV4 Address Pool : ip-addr-pool
  IPv4 DNS Primary : 192.168.10.10
  route accept any tag : 1 distance : 1
```

```
Router#show run all | section ikev2 authorization policy
crypto ikev2 authorization policy default
 route set interface
 route accept any

Router#show crypto ikev2 authorization policy default
 IKEv2 Authorization Policy : default
  route set interface
  route accept any tag : 1 distance : 1
```

```
Router(config)#no crypto ikev2 authorization policy default

Router#show run | include ikev2 authorization policy default
no crypto ikev2 authorization policy default

Router#show crypto ikev2 authorization policy default
 IKEv2 Authorization Policy : default Disabled
```

```
Router(config)#default crypto ikev2 authorization policy
```

```
Router(config)#crypto ikev2 authorization policy default
%Warning: This will Modify Default IKEv2 Authorization Policy. Exit if you don't
want
Router(config-ikev2-author-policy)#pool ip-address-pool
Router(config-ikev2-author-policy)#dns 192.168.6.10

Router#show run | begin ikev2 authorization policy default
crypto ikev2 authorization policy default
 pool ip-address-pool
 dns 192.168.6.10

Router#show crypto ikev2 authorization policy default
 IKEv2 Authorization Policy : default
  IPV4 Address Pool : ip-address-pool
  IPv4 DNS Primary : 192.168.6.10
  route set interface
  route accept any tag : 1 distance : 1
```

```
Router(config)#default crypto ikev2 authorization policy
```

```
Router(config-ikev2-profile)#aaa authorization ?
  group  AAA group authorization
  user   AAA user authorization
```

```
Router(config-ikev2-profile)#aaa authorization user ?
  anyconnect-eap  AAA list to use when IKEv2 remote auth method is anyconnect
                  eap based
  cert            AAA list to use when IKEv2 remote auth method is certificate
                  based
  eap             AAA list to use when IKEv2 remote auth method is EAP
  override        Override user authorization with group authorization. By
                  default, group authorization is overridden with user
                  authorization
  psk             AAA list to use when IKEv2 remote auth method is PSK
```

```
Router(config-ikev2-profile)#aaa authorization user psk ?
  cached  Use cached attributes from EAP authentication or AAA pre-shared key
          fetch
  list    AAA method list

Router(config-ikev2-profile)#aaa authorization user psk cached ?
  <cr>
```

```
Router(config-ikev2-profile)#aaa authorization user psk list ?
  WORD  AAA list name
```

```
Router(config-ikev2-profile)#aaa authorization user psk list aaa_list ?
  WORD          AAA username
  name-mangler  Specify the name-mangler to derive AAA username
  password      Specify the AAA password
  <cr>

Router(config-ikev2-profile)#aaa authorization user psk list aaa_list
name-mangler ?
  WORD  mangler name

Router(config-ikev2-profile)#aaa authorization user psk list aaa_list name-mangle
name_mangler ?
  password  Specify the AAA password
  <cr>
```

```
Router(config-ikev2-profile)# aaa authorization user psk list aaa_list
name-mangler name_mangler password ?
  0      Specifies an UNENCRYPTED password will follow
  6      Specifies an ENCRYPTED password will follow
  WORD   The UNENCRYPTED (cleartext) user password
```

```
aaa group server radius radius_group1
 server name radius_server1

aaa group server radius radius_group2
 server name radius_server2
```

```
aaa authorization network aaa_list1 group radius_group1
aaa authorization network aaa_list2 group radius_group2
```

```
radius server radius_server1
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_server1_key

radius server radius_server2
 address ipv4 172.16.1.4 auth-port 1645 acct-port 1646
 key radius_server2_key
```

```
crypto ikev2 name-mangler name_mangler1
 fqdn hostname


crypto ikev2 name-mangler name_mangler2
 dn common-name
```

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 authentication remote rsa-sig
 aaa authorization user psk list aaa_list1 name-mangler name_mangler1
 aaa authorization user cert list aaa_list2 name-mangler name_mangler2
```

```
IKEv2:(SESSION ID = 96,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]

Initiator SPI : 338E72677B66C73E - Responder SPI : 4C220FBE34587D45 Message id: 1

IKEv2 IKE_AUTH Exchange REQUEST

IKEv2:(SESSION ID = 96,SA ID = 1):Peer's authentication method is 'PSK'

IKEv2:(SESSION ID = 96,SA ID = 1):Searching policy based on peer's identity
'router1.domain1' of type 'FQDN'

IKEv2:(SESSION ID = 96,SA ID = 1):Verification of peer's authenctication data
PASSED
```

```
IKEv2:Using mlist aaa_list1 and username router1 for user author request
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorization request sent
AAA SRV(00000062): Author method=SERVER_GROUP radius_group1
RADIUS/ENCODE(00000062):Orig. component type = VPN IPSEC

RADIUS/ENCODE: Best Local IP-Address 172.16.1.2 for Radius-Server 172.16.1.3
RADIUS(00000062): Send Access-Request to 172.16.1.3:1645 id 1645/85, len 132
RADIUS:  User-Name           [1]   9   "router1"
RADIUS:  User-Password       [2]   18  *
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Service-Type        [6]   6   Outbound                  [5]
RADIUS:  NAS-IP-Address      [4]   6   172.16.1.2
RADIUS(00000062): Sending a IPv4 Radius Packet
RADIUS: Received from id 1645/85 172.16.1.3:1645, Access-Accept, len 128
RADIUS:   Cisco AVpair       [1]   27  "ipsec:route-set=interface"
RADIUS:   Cisco AVpair       [1]   24  "ipsec:route-accept=any"
RADIUS:   Cisco AVpair       [1]   39  "ipsec:route-set=prefix 192.168.1.1/24"
AAA SRV(00000062): protocol reply PASS for Authorization
AAA SRV(00000062): Return Authorization status=PASS
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorization response
```

```
aaa authorization network aaa_list1 local
```

```
crypto ikev2 name-mangler name_mangler1
 dn domain
```

```
crypto ikev2 authorization policy domain1
 route set interface
 route set remote ipv4 192.168.2.0 255.255.255.0
```

```
crypto ikev2 profile default
 match identity remote any
 identity local dn
 authentication local rsa-sig
 authentication remote pre-share
 authentication remote rsa-sig
 pki trustpoint ca_server1
 aaa authorization group cert list aaa_list1 name-mangler name_mangler1
```

```
IKEv2:(SESSION ID = 42,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 91BDE56BC7C68593 - Responder SPI : 0BC460361D49F928 Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST

IKEv2:(SESSION ID = 42,SA ID = 1):Searching policy based on peer's identity 'hostn
ame=router1,cn=router1,dc=domain1' of type 'DER ASN1 DN'

IKEv2:(SESSION ID = 42,SA ID = 1):Peer's authentication method is 'RSA'

IKEv2:(SA ID = 1):[Crypto Engine -> IKEv2] Verification of signed authentication
data PASSED
```

```
IKEv2:Using mlist aaa_list1 and username domain1 for group author request
AAA/AUTHOR (0x21): Pick method list 'aaa_list1'
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorization request sent
AAA SRV(00000021): process author req
AAA SRV(00000021): Author method=LOCAL
AAA/LOCAL/AUTHEN: starting
AAA/LOCAL/AUTHEN(21): authorizing crypto domain1
IKEv2-INTERNAL:IKEv2 local AAA author request for 'domain1'
AAA SRV(00000021): protocol reply PASS for Authorization
AAA SRV(00000021): Return Authorization status=PASS
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorization response
IKEv2-INTERNAL:Received group author attributes:
route-set-interface: 1, route-set-remote-ipv4: 192.168.2.0 255.255.255.0
route-accept any tag:1 distance:1,
```

```
aaa group server radius radius_group1
 server name radius_server1
```

```
aaa authorization network aaa_list1 group radius_group1
```

```
radius server radius_server1
 address ipv4 172.16.1.4 auth-port 1645 acct-port 1646
 key radius_server1_key
```

```
crypto ikev2 name-mangler name_mangler1
 dn domain
```

```
crypto ikev2 profile default
 match identity remote any
 identity local dn
 authentication local rsa-sig
 authentication remote pre-share
 authentication remote rsa-sig
 pki trustpoint ca_server1
 aaa authorization group cert list aaa_list1 name-mangler name_mangler1
```

```
IKEv2:(SESSION ID = 61,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 5FC7B323505CC9F8 - Responder SPI : C8F23357295216E3 Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST

IKEv2:(SESSION ID = 61,SA ID = 1):Searching policy based on peer's identity 'hostn
ame=router1,cn=router1,dc=domain1' of type 'DER ASN1 DN'


IKEv2:(SESSION ID = 61,SA ID = 1):Peer's authentication method is 'RSA'


IKEv2:(SA ID = 1):[Crypto Engine -> IKEv2] Verification of signed authentication
data PASSED
```

```
IKEv2:Using mlist aaa_list1 and username domain1 for group author request
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorization request sent

AAA SRV(00000034): process author req
AAA SRV(00000034): Author method=SERVER_GROUP radius_group1
RADIUS/ENCODE(00000034):Orig. component type = VPN IPSEC
RADIUS/ENCODE: Best Local IP-Address 172.16.1.2 for Radius-Server 172.16.1.4
RADIUS(00000034): Send Access-Request to 172.16.1.4:1645 id 1645/19, len 132
RADIUS:  authenticator BA D7 AE 45 D6 33 76 21 - 20 F8 4F 7A 4C DE E5 1D
RADIUS:  User-Name           [1]   9   "domain1"
RADIUS:  User-Password       [2]   18  *
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  61
RADIUS:   Cisco AVpair       [1]   55  "audit-session-id=L2L4AC1G102ZO2L4AC1G101Z
I1F401F4ZN3D"
RADIUS:  Service-Type        [6]   6   Outbound                  [5]
RADIUS:  NAS-IP-Address      [4]   6   172.16.1.2
RADIUS(00000034): Sending a IPv4 Radius Packet
RADIUS(00000034): Started 5 sec timeout
RADIUS: Received from id 1645/19 172.16.1.4:1645, Access-Accept, len 128
RADIUS:  authenticator EF C6 8B 50 26 AE C2 F3 - 82 2A F1 07 B0 23 C2 ED
RADIUS:  Vendor, Cisco       [26]  33
RADIUS:   Cisco AVpair       [1]   27  "ipsec:route-set=interface"
RADIUS:  Vendor, Cisco       [26]  30
RADIUS:   Cisco AVpair       [1]   24  "ipsec:route-accept=any"
RADIUS:  Vendor, Cisco       [26]  45
RADIUS:   Cisco AVpair       [1]   39  "ipsec:route-set=prefix 192.168.1.1/24"
AAA SRV(00000034): protocol reply PASS for Authorization
AAA SRV(00000034): Return Authorization status=PASS
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response
IKEv2-INTERNAL:Received group author attributes:
route-set-interface: 1, route-accept any tag:1 distance:1,
```

```
aaa group server radius radius_group1
 server name radius_server1
```

```
aaa authorization network aaa_list1 group radius_group1
```

```
radius server radius_server1
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_server1_key
```

```
crypto ikev2 name-mangler name_mangler1
 fqdn hostname
```

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share
 authentication remote pre-share
 keyring aaa aaa_list1 name-mangler name_mangler1
 aaa authorization user psk cached
```

```
IKEv2:(SESSION ID = 3,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]

Initiator SPI : 80753858A6D0BEAB - Responder SPI : EA3464A2957A6F65 Message id: 1

IKEv2 IKE_AUTH Exchange REQUEST

IKEv2:(SESSION ID = 3,SA ID = 1):Searching policy based on peer's identity
'router1.domain1' of type 'FQDN'

IKEv2:(SESSION ID = 3,SA ID = 1):Peer's authentication method is 'PSK'
```

```
IKEv2:(SESSION ID = 3,SA ID = 1):Get peer's preshared key for router1.domain1
IKEv2-INTERNAL:Fetching shared secret from AAA
IKEv2-INTERNAL:Name-mangler 'name_mangler1' returning mangled-name 'router1'
for name-type 1, name-len 15, name_string router1.domain1

IKEv2-INTERNAL:Using AAA method list aaa_list1 for peer router1
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Password request sent

AAA SRV(0000000E): process author req
AAA SRV(0000000E): Author method=SERVER_GROUP radius_group1
RADIUS/ENCODE: Best Local IP-Address 172.16.1.2 for Radius-Server 172.16.1.3
RADIUS(0000000E): Send Access-Request to 172.16.1.3:1645 id 1645/2, len 131
RADIUS:  authenticator 11 DD F2 16 3C 32 7A 9F - 26 6E AA DA 92 8D 99 8D
RADIUS:  User-Name           [1]   9   "router1"
RADIUS:  User-Password       [2]   18  *
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  60
RADIUS:   Cisco AVpair       [1]   54  "audit-session-id=L2L4AC1G102ZO2L4AC1G101Z
I1F401F4ZO3"
RADIUS:  Service-Type        [6]   6   Outbound                  [5]
RADIUS:  NAS-IP-Address      [4]   6   172.16.1.2
RADIUS(0000000E): Sending a IPv4 Radius Packet
RADIUS(0000000E): Started 5 sec timeout
RADIUS: Received from id 1645/2 172.16.1.3:1645, Access-Accept, len 172
RADIUS:  authenticator B0 26 56 33 0F 1D 11 07 - A2 BE 8F 63 DD 71 DB B1
RADIUS:  Vendor, Cisco       [26]  33
RADIUS:   Cisco AVpair       [1]   27  "ipsec:route-set=interface"
RADIUS:  Vendor, Cisco       [26]  30
RADIUS:   Cisco AVpair       [1]   24  "ipsec:route-accept=any"
RADIUS:  Vendor, Cisco       [26]  45
RADIUS:   Cisco AVpair       [1]   39  "ipsec:route-set=prefix 192.168.1.1/24"
RADIUS:  Vendor, Cisco       [26]  44
RADIUS:   Cisco AVpair       [1]   38  "ipsec:tunnel-password=pre_shared_key"
RADIUS(0000000E): Received from id 1645/2
AAA SRV(0000000E): protocol reply PASS for Authorization
AAA SRV(0000000E): Return Authorization status=PASS
```

```
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received password response
IKEv2-INTERNAL:Received symmmetric password from radius server
IKEv2-INTERNAL:Received cached attributes:
route-set-interface: 1, route-accept any tag:1 distance:1,

IKEv2:(SESSION ID = 3,SA ID = 1):Verification of peer's authentication data PASSED
```

```
aaa new-model

aaa group server radius radius_group1
 server name radius_server1

aaa authorization network aaa_psk_list group radius_group1
aaa authorization network user_author_list group radius_group1
aaa authorization network group_author_list local

radius server radius_server1
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_server1_key
```

```
crypto ikev2 name-mangler aaa_psk_name_mangler
 fqdn all

crypto ikev2 name-mangler user_author_name_mangler
 fqdn hostname

crypto ikev2 name-mangler group_author_name_mangler
 fqdn domain
```

```
crypto ikev2 authorization policy domain1
 pool pool3
 def-domain domain1
```

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share
 authentication remote pre-share
 keyring aaa aaa_psk_list name-mangler aaa_psk_name_mangler
 aaa authorization group psk list group_author_list name-mangler
group_author_name_mangler
 aaa authorization user psk cached
 aaa authorization user psk list user_author_list name-mangler
user_author_name_mangler
```

```
IKEv2-INTERNAL:Fetching shared secret from AAA
IKEv2-INTERNAL:Using AAA method list aaa_psk_list for peer router1.domain1
IKEv2-INTERNAL:Received symmmetric password from radius server
IKEv2-INTERNAL:Received cached attributes:
ipv4-pool: pool1, route-set-interface: 1,
```

```
IKEv2:Using mlist group_author_list and username domain1 for group author request
IKEv2-INTERNAL:Received group author attributes:
ipv4-pool: pool3, def-domain: domain1, route-accept any tag:1 distance:1,
```

```
IKEv2:Using mlist user_author_list and username router1 for user author request
IKEv2-INTERNAL:Received user author attributes:
ipv4-pool: pool2, route-accept any tag:1 distance:1,
```

```
IKEv2-INTERNAL:Merging and overriding cached attributes with user attributes
IKEv2-INTERNAL:Merged user attributes:
ipv4-pool: pool2, route-set-interface: 1, route-accept any tag:1 distance:1,
```

```
IKEv2-INTERNAL:Merging and overriding group attributes with user attributes
IKEv2-INTERNAL:Merged attributes:
ipv4-pool: pool2, def-domain: domain1, route-set-interface: 1, route-accept any
tag:1 distance:1,
```

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share
 authentication remote pre-share
 keyring aaa aaa_psk_list name-mangler aaa_psk_name_mangler
 aaa authorization group override psk list group_author_list name-mangler
group_author_name_mangler
 aaa authorization user psk cached
 aaa authorization user psk list user_author_list name-mangler
user_author_name_mangler
```

```
IKEv2-INTERNAL:Fetching shared secret from AAA
IKEv2-INTERNAL:Using AAA method list aaa_psk_list for peer router1.domain1
IKEv2-INTERNAL:Received symmmetric password from radius server
IKEv2-INTERNAL:Received cached attributes:
ipv4-pool: pool1, route-set-interface: 1,
```

```
IKEv2:Using mlist group_author_list and username domain1 for group author request
IKEv2-INTERNAL:Received group author attributes:
ipv4-pool: pool3, def-domain: domain1, route-accept any tag:1 distance:1,
```

```
IKEv2:Using mlist user_author_list and username router1 for user author request
IKEv2-INTERNAL:Received user author attributes:
ipv4-pool: pool2, route-accept any tag:1 distance:1,
```

```
IKEv2-INTERNAL:Merging and overriding cached attributes with user attributes
IKEv2-INTERNAL:Merged user attributes:
ipv4-pool: pool2, route-set-interface: 1, route-accept any tag:1 distance:1,
```

```
IKEv2-INTERNAL:Merging and overriding user attributes with group attributes
IKEv2-INTERNAL:Merged attributes:
ipv4-pool: pool3, def-domain: domain1, route-set-interface: 1, route-accept any
tag:1 distance:1,
```

```
aaa new-model

aaa authorization network default local

crypto ikev2 authorization policy default
 route set interface
 route accept any

crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 aaa authorization group psk list default default
 config-exchange set send
 config-exchange set accept
 config-exchange request
```

```
IKEv2:Config data to send:
IKEv2:(SESSION ID = 1,SA ID = 1):Config-type: Config-request
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-dns, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-dns, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-nbns, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-nbns, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-subnet, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv6-dns, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv6-subnet, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: app-version, length: 277, data:
Cisco IOS Software, Linux Software (I86BI_LINUX-ADVENTERPRISEK9-M), Version
15.6(1.1)T,  ENGINEERING WEEKLY BUILD, synced to  V155_3_M0_2
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2015 by Cisco Systems, Inc.
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: split-dns, length: 0


IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: banner, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: config-url, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: backup-gateway, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: def-domain, length: 0
IKEv2:(SESSION ID = 1,SA ID = 1):Have config mode data to send

IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 172.16.1.2:500/From
172.16.1.1:500/VRF i0:f0]
Initiator SPI : F66257AAE66F66A5 - Responder SPI : 6F177E021F80071F Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):Received Packet [From 172.16.1.2:500/To
172.16.1.1:500/VRF i0:f0]

Initiator SPI : F66257AAE66F66A5 - Responder SPI : 6F177E021F80071F Message id: 1

IKEv2 IKE_AUTH Exchange RESPONSE


IKEv2:(SESSION ID = 1,SA ID = 1):Verification of peer's authenctication data
PASSED

IKEv2-INTERNAL:IKEv2 local AAA author request for 'default'

IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response

IKEv2-INTERNAL:VPN route info added for 'route set interface' attribute

IKEv2-INTERNAL:Received group author attributes:

route-set-interface: 1, route-accept any tag:1 distance:1,


IKEv2:Config data received:

IKEv2:(SESSION ID = 1,SA ID = 1):Config-type: Config-reply

IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-subnet, length: 8, data:
10.0.0.2 255.255.255.255

IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: app-version, length: 277, data:
Cisco IOS Software, Linux Software (I86BI_LINUX-ADVENTERPRISEK9-M), Version
15.6(1.1)T,  ENGINEERING WEEKLY BUILD, synced to  V155_3_M0_2

Technical Support: http://www.cisco.com/techsupport

Copyright (c) 1986-2015 by Cisco Systems, Inc.
```

```
IKEv2-INTERNAL:Sending config-set
IKEv2:(SESSION ID = 1,SA ID = 1):Config-type: Config-set
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-subnet, length: 8, data:
10.0.0.1 255.255.255.255
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: app-version, length: 277, data:
Cisco IOS Software, Linux Software (I86BI_LINUX-ADVENTERPRISEK9-M), Version
15.6(1.1)T,  ENGINEERING WEEKLY BUILD, synced to  V155_3_M0_2
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2015 by Cisco Systems, Inc.

IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 172.16.1.2:500/From
172.16.1.1:500/VRF i0:f0]
Initiator SPI : F66257AAE66F66A5 - Responder SPI : 6F177E021F80071F Message id: 2
IKEv2 INFORMATIONAL Exchange REQUEST
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):Received Packet [From 172.16.1.2:500/To
172.16.1.1:500/VRF i0:f0]
Initiator SPI : F66257AAE66F66A5 - Responder SPI : 6F177E021F80071F Message id: 2
IKEv2 INFORMATIONAL Exchange RESPONSE

IKEv2:Config data received:
IKEv2:(SESSION ID = 1,SA ID = 1):Config-type: Config-ack
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-subnet, length: 0
```

```
IKEv2:(SESSION ID = 8,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 165DFC7D8B4DDE0A - Responder SPI : 61B417D0A611857A Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST

IKEv2:[Crypto Engine -> IKEv2] IKEv2 authentication data generation PASSED
IKEv2:Using mlist default and username default for group author request
IKEv2-INTERNAL:Received group author attributes:
route-set-interface: 1, route-accept any tag:1 distance:1,

IKEv2:Config data received:
IKEv2:(SESSION ID = 8,SA ID = 1):Config-type: Config-request
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-dns, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-dns, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-nbns, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-nbns, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-subnet, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv6-dns, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv6-subnet, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: app-version, length: 277, data:
Cisco IOS Software, Linux Software (I86BI_LINUX-ADVENTERPRISEK9-M), Version
15.6(1.1)T,  ENGINEERING WEEKLY BUILD, synced to  V155_3_M0_2
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2015 by Cisco Systems, Inc.
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: split-dns, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: banner, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: config-url, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: backup-gateway, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: def-domain, length: 0
IKEv2:(SESSION ID = 8,SA ID = 1):Set received config mode data
```

```
IKEv2:Config data to send:
IKEv2:(SESSION ID = 8,SA ID = 1):Config-type: Config-reply
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-subnet, length: 8, data:
10.0.0.2 255.255.255.255
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: app-version, length: 277, data:
Cisco IOS Software, Linux Software (I86BI_LINUX-ADVENTERPRISEK9-M), Version
15.6(1.1)T,  ENGINEERING WEEKLY BUILD, synced to  V155_3_M0_2
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2015 by Cisco Systems, Inc.

IKEv2:(SESSION ID = 8,SA ID = 1):Sending Packet [To 172.16.1.1:500/From
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 165DFC7D8B4DDE0A - Responder SPI : 61B417D0A611857A Message id: 1
IKEv2 IKE_AUTH Exchange RESPONSE
```

IKEv2:(SESSION ID = 8,SA ID = 1):Received Packet [From 172.16.1.1:500/To 172.16.1.2:500/VRF i0:f0]

Initiator SPI : 165DFC7D8B4DDE0A - Responder SPI : 61B417D0A611857A Message id: 2

IKEv2 INFORMATIONAL Exchange REQUEST

IKEv2:Config data received:

IKEv2:(SESSION ID = 8,SA ID = 1):Config-type: Config-set

IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-subnet, length: 8, data: 10.0.0.1 255.255.255.255

IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: app-version, length: 277, data: Cisco IOS Software, Linux Software (I86BI_LINUX-ADVENTERPRISEK9-M), Version 15.6(1.1)T, ENGINEERING WEEKLY BUILD, synced to V155_3_M0_2

Technical Support: http://www.cisco.com/techsupport

Copyright (c) 1986-2015 by Cisco Systems, Inc.

```
IKEv2:Config data to send:
IKEv2:(SESSION ID = 8,SA ID = 1):Config-type: Config-ack
IKEv2:(SESSION ID = 8,SA ID = 1):Attrib type: ipv4-subnet, length: 0


IKEv2:(SESSION ID = 8,SA ID = 1):Sending Packet [To 172.16.1.1:500/From
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 165DFC7D8B4DDE0A - Responder SPI : 61B417D0A611857A Message id: 2
IKEv2 INFORMATIONAL Exchange RESPONSE
```

```
crypto ikev2 profile default
 match identity remote any
 identity local fqdn router1.domain1
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 aaa authorization group psk list default default
 no config-exchange request
```

```
IKEv2:(SESSION ID = 9,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]

Initiator SPI : 789338A336F1ACC2 - Responder SPI : 599AB29DACDBC6B5 Message id: 1

IKEv2 IKE_AUTH Exchange REQUEST


IKEv2:(SESSION ID = 9,SA ID = 1):Verification of peer's authenctication data
PASSED

IKEv2:Using mlist default and username default for group author request

IKEv2-INTERNAL:Received group author attributes:

route-set-interface: 1, route-accept any tag:1 distance:1,
```

```
IKEv2-INTERNAL:Sending config-set
IKEv2:(SESSION ID = 9,SA ID = 1):Config-type: Config-set
IKEv2:(SESSION ID = 9,SA ID = 1):Attrib type: ipv4-subnet, length: 8, data:
10.0.0.2 255.255.255.255
IKEv2:(SESSION ID = 9,SA ID = 1):Attrib type: app-version, length: 277, data:
Cisco IOS Software, Linux Software (I86BI_LINUX-ADVENTERPRISEK9-M), Version
15.6(1.1)T,  ENGINEERING WEEKLY BUILD, synced to  V155_3_M0_2
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2015 by Cisco Systems, Inc.


IKEv2:(SESSION ID = 9,SA ID = 1):Sending Packet [To 172.16.1.1:500/From
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 789338A336F1ACC2 - Responder SPI : 599AB29DACDBC6B5 Message id: 0
IKEv2 INFORMATIONAL Exchange REQUEST
```

```
IKEv2:(SESSION ID = 9,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 789338A336F1ACC2 - Responder SPI : 599AB29DACDBC6B5 Message id: 0
IKEv2 INFORMATIONAL Exchange RESPONSE

IKEv2:Config data received:
IKEv2:(SESSION ID = 9,SA ID = 1):Config-type: Config-ack
IKEv2:(SESSION ID = 9,SA ID = 1):Attrib type: ipv4-subnet, length: 0
```

```
aaa new-model

aaa authorization network default local

crypto ikev2 authorization policy default
 route set local ipv4 192.168.6.0 255.255.255.0 tag 5000 distance 5
 route set interface

crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 aaa authorization group psk list default default
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):Received Packet [From 172.16.1.2:500/To
172.16.1.1:500/VRF i0:f0]
Initiator SPI : 30DCC2A72A9F875A - Responder SPI : E2FB7F51411F1A5E Message id: 1
IKEv2 IKE_AUTH Exchange RESPONSE

IKEv2:(SESSION ID = 1,SA ID = 1):Verification of peer's authentication data PASSED
IKEv2-INTERNAL:IKEv2 local AAA author request for 'default'
IKEv2-INTERNAL:Received group author attributes:
route-set-interface: 1, route-set-local-ipv4: 192.168.6.0 255.255.255.0 tag 5000
distance 5 route-accept any tag:1 distance:1,
```

```
IKEv2-INTERNAL:VPN Route Added 192.168.6.0 255.255.255.0 via Tunnel0 in vrf global
```

```
Router#show ip route 192.168.6.0
Routing entry for 192.168.6.0/24
  Known via "static", distance 5, metric 0 (connected)
  Tag 5000
  Routing Descriptor Blocks:
  * directly connected, via Tunnel0, permanent
      Route metric is 0, traffic share count is 1
      Route tag 5000
```

```
aaa new-model

aaa authorization network default local

crypto ikev2 authorization policy default
 no route set interface
 route accept any

crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 aaa authorization group psk list default default
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):Received Packet [From 172.16.1.2:500/To
172.16.1.1:500/VRF i0:f0]
Initiator SPI : AE84121BC64D641B - Responder SPI : 89F6DA91AC2A2823 Message id: 1
IKEv2 IKE_AUTH Exchange RESPONSE

IKEv2:(SESSION ID = 1,SA ID = 1):Verification of peer's authentication data PASSED
IKEv2:Using mlist default and username default for group author request
IKEv2-INTERNAL:Received group author attributes:
route-accept any tag:1 distance:1,
```

```
IKEv2-INTERNAL:Received config data from toolkit:
IKEv2:Config data received:
IKEv2:(SESSION ID = 1,SA ID = 1):Config-type: Config-reply
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: ipv4-subnet, length: 8, data:
10.0.0.2 255.255.255.255
```

```
IKEv2-INTERNAL:Subnet address: 10.0.0.2 255.255.255.255
IKEv2:VPN Route Added 10.0.0.2 255.255.255.255 via Tunnel0 in vrf global
```

```
Router#show ip route
Gateway of last resort is not set

      4.0.0.0/24 is subnetted, 1 subnets
S        4.4.4.0 is directly connected, Tunnel0
      10.0.0.0/8 is variably subnetted, 3 subnets, 2 masks
S        10.0.0.2/32 is directly connected, Tunnel0

Router#show ip route 10.0.0.2
Routing entry for 10.0.0.2/32
  Known via "static", distance 1, metric 0 (connected)
  Tag 1
  Routing Descriptor Blocks:
  * directly connected, via Tunnel0, permanent
      Route metric is 0, traffic share count is 1
      Route tag 1
```

```
Router(config-ikev2-profile)#authentication remote eap
For remote auth method to be EAP, the local auth method must be certificate-based
Router(config-ikev2-profile)#authentication local rsa-sig
Router(config-ikev2-profile)#authentication remote eap
```

```
Router(config-ikev2-profile)#authentication remote eap ?
  query-identity  query EAP identity from peer
  timeout         timeout
  <cr>
```

```
Router(config-ikev2-profile)#authentication remote eap timeout ?
  <45-180>  EAP authentication timeout in seconds
```

```
Router(config-ikev2-profile)#aaa authentication eap ?
  WORD  AAA list name
```

```
aaa authentication login authen_list_radius group radius_group
```

```
aaa group server radius radius_group
 server name radius_server
```

```
radius server radius_server
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_key
```

```
crypto ikev2 profile ikev2_profile
 aaa authentication eap authen_list_radius
 authentication remote eap query-identity
 authentication remote eap timeout 120
 authentication remote rsa-sig
 aaa authentication eap authen_list_radius
```

```
IKEv2:(SESSION ID = 69,SA ID = 1):Received Packet [From 172.16.1.1:59040/To
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST
Payload contents:
IKEv2-INTERNAL:Parse Vendor Specific Payload: (CUSTOM) VID IDi CERTREQ CFG SA
```

```
IKEv2:(SESSION ID = 69,SA ID = 1):Generating EAP request
IKEv2:(SESSION ID = 69,SA ID = 1):Building packet for encryption.
Payload contents:
 VID IDr CERT CERT AUTH EAP
IKEv2:(SESSION ID = 69,SA ID = 1):Sending Packet [To 172.16.1.1:59040/From
172.16.1.2:4500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 1
IKEv2 IKE_AUTH Exchange RESPONSE
```

```
IKEv2:(SESSION ID = 69,SA ID = 1):Received Packet [From 172.16.1.1:59040/To
172.16.1.2:4500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 2
IKEv2 IKE_AUTH Exchange REQUEST
Payload contents:
 EAP
IKEv2:(SESSION ID = 69,SA ID = 1):Processing EAP response
IKEv2:Use authen method list TEST
IKEv2:pre-AAA: client sent User1 as EAP-Id response
```

```
IKEv2:sending User1 [EAP-Id] as username to AAA
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authentication request sent
RADIUS(00000049): Send Access-Request to 172.16.1.3:1812 id 1645/36, len 196
RADIUS:  authenticator 5A 04 CC CF DA 86 4C D3 - 4E B8 8A F3 75 47 52 81
RADIUS:  Service-Type          [6]   6   Login                        [1]
RADIUS:  Vendor, Cisco         [26]  26
RADIUS:   Cisco AVpair         [1]   20  "service-type=Login"
RADIUS:  Vendor, Cisco         [26]  28
RADIUS:   Cisco AVpair         [1]   22  "isakmp-phase1-id=router1"
RADIUS:  Calling-Station-Id    [31]  16  "172.16.1.2"
RADIUS:  Vendor, Cisco         [26]  63
RADIUS:   Cisco AVpair         [1]   57  "audit-session-
id=L2L40A683111ZO2L40A6869A2ZH1194E6AZO45"
RADIUS:  User-Name             [1]   4   "User1"
RADIUS:  EAP-Message           [79]  9
RADIUS:   02 3B 00 07 01 70 31                [ ;User1]
RADIUS:  Message-Authenticato[80]  18
RADIUS:   41 41 1E 2B 3B B9 4A 55 F0 1C 04 0A D4 60 62 86          [ AA+;JU`b]
RADIUS:  NAS-IP-Address        [4]   6   10.104.49.17
```

```
RADIUS: Received from id 1645/36 172.16.1.3:1812, Access-Challenge, len 1159
RADIUS:  authenticator 52 73 15 7B 6F A2 46 38 - 7D 6B EB 3D 81 88 71 D0
RADIUS:  Service-Type          [6]   6   NAS Prompt                [7]
RADIUS:  EAP-Message           [79]  24

IKEv2:(SESSION ID = 69,SA ID = 1):Generating EAP request
IKEv2:(SESSION ID = 69,SA ID = 1):Building packet for encryption.
Payload contents:
 EAP
IKEv2:(SESSION ID = 69,SA ID = 1):Sending Packet [To 172.16.1.1:59040/From
172.16.1.2:4500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 2
IKEv2 IKE_AUTH Exchange RESPONSE
```

```
IKEv2:(SESSION ID = 69,SA ID = 1):Received Packet [From 172.16.1.1:59040/To
172.16.1.2:4500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 3
IKEv2 IKE_AUTH Exchange REQUEST
Payload contents:
 EAP

RADIUS(00000049): Send Access-Request to 172.16.1.3:1812 id 1645/37, len 229
RADIUS:  authenticator 3B 58 59 F2 D3 AC F8 18 - 5C E8 6C C2 A9 BD 1B B9
RADIUS:  Service-Type          [6]   6   Login                        [1]
RADIUS:  Vendor, Cisco         [26]  26
RADIUS:   Cisco AVpair         [1]   20  "service-type=Login"
RADIUS:  Vendor, Cisco         [26]  28
RADIUS:   Cisco AVpair         [1]   22  "isakmp-phase1-id=router1"
RADIUS:  Calling-Station-Id    [31]  16  "172.16.1.2"
RADIUS:  Vendor, Cisco         [26]  63
RADIUS:   Cisco AVpair         [1]   57  "audit-session-
id=L2L40A683111ZO2L40A6869A2ZH1194E6AZO45"
RADIUS:  User-Name             [1]   4   "User1"
RADIUS:  EAP-Message           [79]  24
```

```
RADIUS: Received from id 1645/37 172.16.1.3:1812, Access-Accept, len 1127
RADIUS:  authenticator 7B 7A B6 8B 64 E1 2D 03 - 86 74 B9 21 39 A7 1B FB
RADIUS:  Service-Type          [6]   6   NAS Prompt                [7]
RADIUS:  EAP-Message           [79]  6

IKEv2:(SESSION ID = 69,SA ID = 1):Sending EAP status message
IKEv2:(SESSION ID = 69,SA ID = 1):Building packet for encryption.
Payload contents:
 EAP
IKEv2:(SESSION ID = 69,SA ID = 1):Sending Packet [To 172.16.1.1:59040/From
172.16.1.2:4500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 3
IKEv2 IKE_AUTH Exchange RESPONSE
```

```
IKEv2:(SESSION ID = 69,SA ID = 1):Received Packet [From 172.16.1.1:59040/To
172.16.1.2:4500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 4
IKEv2 IKE_AUTH Exchange REQUEST
Payload contents:
 AUTH
```

```
IKEv2:(SESSION ID = 69,SA ID = 1):Building packet for encryption.
Payload contents:
 AUTH CFG SA TSi TSr NOTIFY(SET_WINDOW_SIZE) NOTIFY(ESP_TFC_NO_SUPPORT)
NOTIFY(NON_FIRST_FRAGS)
IKEv2:(SESSION ID = 69,SA ID = 1):Sending Packet [To 172.16.1.1:59040/From
172.16.1.2:4500/VRF i0:f0]
Initiator SPI : 3A1C6331D3A4726E - Responder SPI : 123A08C78505C625 Message id: 4
IKEv2 IKE_AUTH Exchange RESPONSE
Payload contents:
 ENCR
```

```
Router#show crypto ikev2 sa detail
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         172.16.1.2/4500       172.16.1.2/59040      none/none             READY
      Encr: 3DES, PRF: SHA1, Hash: SHA96, DH Grp:2, Auth sign: RSA, Auth verify: EAP
      Life/Active Time: 900/21 sec
      CE id: 1028, Session-id: 22
      Status Description: Negotiation done
      Local spi: 123A08C78505C625       Remote spi: 3A1C6331D3A4726E
      Local id: 172.16.1.2
      Remote id: router1
      Remote EAP id: User1
```

```
<ServerList>
  <HostEntry>
   <HostName>vpn1.cisco.com</HostName>
   <HostAddress>vpn1.cisco.com</HostAddress>
      <PrimaryProtocol>IPsec
      <StandardAuthenticationOnly>true
      <AuthMethodDuringIKENegotiation>EAP-MSCHAPv2</
AuthMethodDuringIKENegotiation>
      <IKEIdentity>cisco.com</IKEIdentity>
      </StandardAuthenticationOnly>
  </PrimaryProtocol>
  </HostEntry>
 </ServerList>
```

```
<ServerList>
  <HostEntry>
   <HostName>vpn2.cisco.com</HostName>
   <HostAddress>vpn2.cisco.com</HostAddress>
      <PrimaryProtocol>IPsec
      <StandardAuthenticationOnly>true
      <AuthMethodDuringIKENegotiation>EAP-MD5</AuthMethodDuringIKENegotiation>
      <IKEIdentity>cisco.com</IKEIdentity>
      </StandardAuthenticationOnly>
  </PrimaryProtocol>
  </HostEntry>
 </ServerList>
```

```
Router(config-ikev2-profile)#keyring ?
  aaa     AAA-based keyring
  local   Local keyring
```

```
Router(config-ikev2-profile)#keyring aaa ?
  WORD  AAA list name
```

```
Router(config-ikev2-profile)#keyring aaa aaa_psk_list ?
  name-mangler  Specify the name-mangler to derive username
  password      Specify the AAA password
  <cr>
```

```
aaa authorization network aaa_list1 group radius_group1
```

```
aaa group server radius radius_group
 server name radius_server
```

```
radius server radius_server
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_key
```

```
crypto ikev2 name-mangler name_mangler1
 fqdn hostname
```

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share
 authentication remote pre-share
 keyring aaa aaa_list1 name-mangler name_mangler1
 aaa authorization user psk cached
```

```
IKEv2:(SESSION ID = 3,SA ID = 1):Received Packet [From 172.16.1.1:500/To
172.16.1.2:500/VRF i0:f0]

Initiator SPI : 80753858A6D0BEAB - Responder SPI : EA3464A2957A6F65 Message id: 1

IKEv2 IKE_AUTH Exchange REQUEST

IKEv2:(SESSION ID = 3,SA ID = 1):Searching policy based on peer's identity
'router1.domain1' of type 'FQDN'

IKEv2:(SESSION ID = 3,SA ID = 1):Peer's authentication method is 'PSK'
```

```
IKEv2:(SESSION ID = 3,SA ID = 1):Get peer's preshared key for router1.domain1
IKEv2-INTERNAL:Fetching shared secret from AAA
IKEv2-INTERNAL:Name-mangler 'name_mangler1' returning mangled-name 'router1'
for name-type 1, name-len 15, name_string router1.domain1

IKEv2-INTERNAL: Using AAA method list aaa_list1 for peer router1
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Password request sent

AAA SRV(0000000E): process author req
AAA SRV(0000000E): Author method=SERVER_GROUP radius_group1
RADIUS/ENCODE: Best Local IP-Address 172.16.1.2 for Radius-Server 172.16.1.3
RADIUS(0000000E): Send Access-Request to 172.16.1.3:1645 id 1645/2, len 131
RADIUS:  authenticator 11 DD F2 16 3C 32 7A 9F - 26 6E AA DA 92 8D 99 8D
RADIUS:  User-Name           [1]   9   "router1"
RADIUS:  User-Password       [2]   18  *
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  60
RADIUS:   Cisco AVpair       [1]   54  "audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZO3"
RADIUS:  Service-Type        [6]   6   Outbound                    [5]
RADIUS:  NAS-IP-Address      [4]   6   172.16.1.2
RADIUS(0000000E): Sending a IPv4 Radius Packet
RADIUS(0000000E): Started 5 sec timeout
```

```
RADIUS: Received from id 1645/2 172.16.1.3:1645, Access-Accept, len 172
RADIUS:  authenticator B0 26 56 33 0F 1D 11 07 - A2 BE 8F 63 DD 71 DB B1
RADIUS:  Vendor, Cisco       [26]  33
RADIUS:   Cisco AVpair       [1]   27  "ipsec:route-set=interface"
RADIUS:  Vendor, Cisco       [26]  30
RADIUS:   Cisco AVpair       [1]   24  "ipsec:route-accept=any"
RADIUS:  Vendor, Cisco       [26]  45
RADIUS:   Cisco AVpair       [1]   39  "ipsec:route-set=prefix 192.168.1.1/24"
RADIUS:  Vendor, Cisco       [26]  44
RADIUS:   Cisco AVpair       [1]   38  "ipsec:tunnel-password=pre_shared_key"
RADIUS(0000000E): Received from id 1645/2
AAA SRV(0000000E): protocol reply PASS for Authorization
AAA SRV(0000000E): Return Authorization status=PASS
```

```
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received password response
IKEv2-INTERNAL:Received symmetric password from radius server
IKEv2-INTERNAL:Received cached attributes:
route-set-interface: 1, route-accept any tag:1 distance:1,

IKEv2:(SESSION ID = 3,SA ID = 1):Verification of peer's authentication data PASSED
```

```
Router(config-ikev2-profile)#aaa accounting ?
  anyconnect-eap  AAA list to use when IKEv2 remote auth method is AnyConnect
                  EAP
  cert            AAA list to use when IKEv2 remote auth method is certificate
                  based
  eap             AAA list to use when IKEv2 remote auth method is EAP
  psk             AAA list to use when IKEv2 remote auth method is PSK
```

```
Router(config-ikev2-profile)#aaa accounting psk ?
  WORD  AAA list name
```

```
aaa accounting network acc_list
 action-type start-stop
 group radius_group1
```

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 aaa accounting psk acc_list
```

```
IKEv2:(SESSION ID = 8,SA ID = 1):Sending Packet [To 172.16.1.1:500/From
172.16.1.2:500/VRF i0:f0]
Initiator SPI : 0A40B2679CC0B5C2 - Responder SPI : 6833E924871B8C81 Message id: 1
IKEv2 IKE_AUTH Exchange RESPONSE
Payload contents:
 ENCR

IKEv2:(SESSION ID = 8,SA ID = 1):IKEV2 SA created; inserting SA into database. SA
lifetime timer (86400 sec) started
IKEv2:(SA ID = 1):[IPsec -> IKEv2] Creation of IPsec SA into IPsec database PASSED
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up
AAA/ACCT/EVENT/(00000011): CALL START
AAA/ACCT/EVENT/(00000011): IPSEC TNL UP
AAA/ACCT/IPSEC-TUNNEL(00000011): Queueing record is START

IKEv2:(SA ID = 1):[IKEv2 -> AAA] Accounting start request sent successfully
RADIUS(00000011): Send Accounting-Request to 172.16.1.3:1646 id 1646/9, len 256
RADIUS:  authenticator 6A 7A 8F EA 4E 10 E9 CE - 8C C9 EC A8 C4 70 C7 9F
RADIUS:  Acct-Session-Id      [44]  10  "00000007"
RADIUS:  Calling-Station-Id   [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco        [26]  60
RADIUS:   Cisco AVpair        [1]   54  "audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZO8"
RADIUS:  Vendor, Cisco        [26]  40
RADIUS:   Cisco AVpair        [1]   34  "isakmp-phase1-id=router1.domain1"
RADIUS:  Vendor, Cisco        [26]  37
RADIUS:   Cisco AVpair        [1]   31  "isakmp-initator-ip=172.16.1.1"
RADIUS:  User-Name            [1]   17  "router1.domain1"
RADIUS:  Vendor, Cisco        [26]  36
RADIUS:   Cisco AVpair        [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Authentic       [45]  6   Local                       [2]
RADIUS:  Acct-Status-Type     [40]  6   Start                       [1]
RADIUS:  NAS-IP-Address       [4]   6   172.16.1.2
RADIUS:  Acct-Delay-Time      [41]  6   0
```

```
AAA/ACCT/EVENT/(00000011): IPSEC TNL DOWN
IKEv2:in_octets 13020, out_octets 17640
IKEv2:in_packets 105, out_packets 105
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Accounting stop request sent successfully
AAA/ACCT/EVENT/(00000011): CALL STOP
AAA/ACCT/IPSEC-TUNNEL(00000011): STOP

RADIUS(00000011): Send Accounting-Request to 172.16.1.3:1646
id 1646/10, len 324
RADIUS:  authenticator 6E 18 B4 4F E0 B5 C0 09 - 9F B9 FC 01 5A D4 69 6F
RADIUS:  Acct-Session-Id      [44]  10  "00000007"
RADIUS:  Calling-Station-Id   [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco        [26]  60
RADIUS:   Cisco AVpair        [1]   54  "audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZO8"
RADIUS:  Vendor, Cisco        [26]  40
RADIUS:   Cisco AVpair        [1]   34  "isakmp-phase1-id=router1.domain1"
RADIUS:  Vendor, Cisco        [26]  37
RADIUS:   Cisco AVpair        [1]   31  "isakmp-initator-ip=172.16.1.1"
RADIUS:  User-Name            [1]   17  "router1.domain1"
RADIUS:  Acct-Authentic       [45]  6   Local                       [2]
RADIUS:  Vendor, Cisco        [26]  36
RADIUS:   Cisco AVpair        [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Session-Time    [46]  6   34
RADIUS:  Acct-Input-Octets    [42]  6   13020
RADIUS:  Acct-Output-Octets   [43]  6   17640
RADIUS:  Acct-Input-Packets   [47]  6   105
RADIUS:  Acct-Output-Packets  [48]  6   105
RADIUS:  Acct-Terminate-Cause[49]  6   none                        [0]
RADIUS:  Vendor, Cisco        [26]  32
RADIUS:   Cisco AVpair        [1]   26  "disc-cause-ext=No Reason"
RADIUS:  Acct-Status-Type     [40]  6   Stop                        [2]
RADIUS:  NAS-IP-Address       [4]   6   172.16.1.2
RADIUS:  Acct-Delay-Time      [41]  6   0
```

```
Router(config)#interface virtual-template ?
  <1-200>  Virtual-Template interface number

Router(config)#interface virtual-template 1 ?
  type  type of the virtual-template
  <cr>

Router(config)#interface virtual-template 1 type ?
  ethernet  Set VT type as ethernet
  serial    Set VT type as serial
  tunnel    Set VT type as tunnel
```

```
Router(config)#interface virtual-template 1 type tunnel ?
  <cr>
```

```
Router(config)#interface virtual-template 1 type tunnel
Router(config-if)#ip unnumbered e0/0
Router(config-if)#tunnel protection ipsec profile default
```

```
Router(config-ikev2-profile)#virtual-template 1
```

```
Router#show vtemplate 1
Virtual-access subinterface creation is globally enabled

          Active     Active    Subint  Interface
        Interface Subinterface Capable Type
        --------- ------------ ------- ---------
Vt1             1            0    No   Tunnel

Router(config)#interface virtual-template 1 type tunnel
% Virtual-template config is locked, active vaccess present
Router(config)#
```

```
Router#show vtemplate 1
Virtual-access subinterface creation is globally enabled

          Active     Active    Subint  Interface
        Interface Subinterface Capable Type
        --------- ------------ ------- ---------
Vt1             0            0    No   Tunnel

Router(config)#interface virtual-template 1 type tunnel
Router(config-if)#
```

```
aaa authorization network default local
aaa authorization network aaa_list1 group radius_group1
```

```
crypto ikev2 name-mangler name_mangler1
 fqdn hostname

crypto ikev2 name-mangler name_mangler2
 fqdn domain
```

```
aaa attribute list attr_list1
 attribute type interface-config "tunnel key 1"
 attribute type interface-config "ip mtu 1000"

crypto ikev2 authorization policy domain1
 aaa attribute list attr_list1
```

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 aaa authorization group psk list default name-mangler name_mangler2
 aaa authorization user psk list aaa_list1 name-mangler name_mangler1
 virtual-template 1
```

```
interface Virtual-Template1 type tunnel
 ip unnumbered Ethernet0/0
 tunnel protection ipsec profile default
```

```
VT[Vi1]:Clone Vaccess from Virtual-Template1 (45 bytes)
VT[Vi1]:tunnel protection ipsec profile default
VT[Vi1]:end
```

```
INFO: AAA/AUTHOR: Processing PerUser AV interface-config
INFO: AAA/AUTHOR: Processing PerUser AV interface-config
VT[Vi1]:Clone Vaccess from AAA (30 bytes)
VT[Vi1]:tunnel key 1
VT[Vi1]:ip mtu 1000
VT[Vi1]:end
```

```
INFO: AAA/AUTHOR: Processing PerUser AV interface-config
INFO: AAA/AUTHOR: Processing PerUser AV interface-config
VT[Vi1]:Clone Vaccess from AAA (30 bytes)
VT[Vi1]:ip mtu 1200
VT[Vi1]:tunnel key 2
VT[Vi1]:end
```

```
Router2#show crypto session
Crypto session current status
Interface: Virtual-Access1
Profile: default
Session status: UP-ACTIVE
Peer: 172.16.1.1 port 500
  Session ID: 28
  IKEv2 SA: local 172.16.1.2/500 remote 172.16.1.1/500 Active
  IPSEC FLOW: permit 47 host 172.16.1.2 host 172.16.1.1
        Active SAs: 2, origin: crypto-map
```

```
Router2#show derived-config interface virtual-access 1
Building configuration...

Derived configuration : 222 bytes

interface Virtual-Access1
 ip unnumbered Ethernet0/0
 ip mtu 1200
 tunnel source 172.16.1.2
 tunnel destination 172.16.1.1
 tunnel key 2
 tunnel protection ipsec profile default
 no tunnel protection ipsec initiate
end
```

```
crypto ikev2 profile default
 match identity remote any
 authentication local pre-share key pre_shared_key
 authentication remote pre-share key pre_shared_key
 virtual-template 1 mode auto
```

```
Router#show interfaces virtual-template 1 | inc protocol/transport
  Tunnel protocol/transport GRE/IP
```

```
IPSEC(validate_proposal_request): proposal part #1,
  (key eng. msg.) INBOUND local= 172.16.1.2:0, remote= 172.16.1.1:0,
    local_proxy= 0.0.0.0/0.0.0.0/256/0,
    remote_proxy= 0.0.0.0/0.0.0.0/256/0,
    protocol= ESP, transform= esp-aes esp-sha-hmac  (Tunnel),
```

```
Router#show interfaces virtual-access 1 | inc protocol/transport
  Tunnel protocol/transport IPSEC/IP
```

```
Router#show interfaces virtual-template 1 | inc protocol/transport
  Tunnel protocol/transport IPSEC/IP
```

```
IPSEC(validate_proposal_request): proposal part #1,
  (key eng. msg.) INBOUND local= 172.16.1.2:0, remote= 172.16.1.1:0,
    local_proxy= 172.16.1.2/255.255.255.255/47/0,
    remote_proxy= 172.16.1.1/255.255.255.255/47/0,
    protocol= ESP, transform= esp-aes esp-sha-hmac  (Transport),

Router#show interfaces virtual-access 1 | inc protocol/transport
  Tunnel protocol/transport GRE/IP
```

```
Router(config)#aaa server radius ?
  dynamic-author  Local server profile for RFC 3576 support
Router(config)#aaa server radius dynamic-author
Router(config-locsvr-da-radius)#?
RADIUS Application commands:
  client      Specify a RADIUS client
  port        Specify port on which local radius server listens
  server-key  Encryption key shared with the radius clients
```

```
Router(config-locsvr-da-radius)#client ?
  Hostname or A.B.C.D  IP address of RADIUS client
  X:X:X:X::X           IPv6 address of RADIUS client
```

```
Router(config-locsvr-da-radius)#server-key ?
  0      Specifies an UNENCRYPTED key will follow
  6      Specifies an ENCRYPTED key will follow
  7      Specifies HIDDEN key will follow
  WORD   The UNENCRYPTED (cleartext) shared key
```

```
IKEv2:Using mlist aaa_list1 and username router1 for user author request
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorization request sent
RADIUS(000000D0): Send Access-Request to 172.16.1.3:1645 id 1645/37, len 132
RADIUS:  authenticator E0 93 9F 13 28 EB 8A EA - 64 46 2B 52 53 CF D1 33
RADIUS:  User-Name           [1]   9   "router1"
RADIUS:  User-Password       [2]   18  *
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  61
RADIUS:   Cisco AVpair       [1]   55  "audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN34"
```

```
RADIUS(000000D3): Send Accounting-Request to 172.16.1.3:1646 id 1646/96, len 267
RADIUS:  authenticator DD 1D 68 F4 B0 EF 11 DA - 91 22 C2 9E 7C 86 5F A7
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  61
RADIUS:   Cisco AVpair       [1]   55  "audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN34"
```

```
Router2#show crypto ikev2 sa detail
 IPv4 Crypto IKEv2  SA
Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         172.16.1.2/500        172.16.1.1/500        none/none            READY
      Encr: AES-CBC, keysize: 256, PRF: SHA512, Hash: SHA512, DH Grp:5, Auth sign:
PSK, Auth verify: PSK
      Life/Active Time: 86400/3 sec
      CE id: 1057, Session-id: 52
      Status Description: Negotiation done
      Local spi: 8D331A95A67688AB       Remote spi: E7286487AC534733
      Local id: 172.16.1.2
      Remote id: router1.domain1
```

```
Router#show aaa user all | begin router1.domain1
  IPSEC-TUNNEL: Username=router1.domain1
    Session Id=000000CE Unique Id=000000D8
    Start Sent=1 Stop Only=N
    stop_has_been_sent=N
    Method List=8655B740 : Name = acc_list
    Attribute list:
      86D5EEF8 0 00000082 formatted-clid(37) 10 172.16.1.1
      86D5EF2C 0 0000008A audit-session-id(819) 36
L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN35
      86D5EF60 0 00000081 isakmp-phase1-id(737) 15 router1.domain1
      86D5EF94 0 00000002 isakmp-initator-ip(738) 4 172.16.1.1
```

```
RADIUS: POD  received from id 6 172.16.1.3:1700, POD Request, len 84
POD: 172.16.1.3 request queued
 ++++++ POD Attribute List ++++++
86D5EDB4 0 00000089 audit-session-id(819) 36
L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN34
```

POD: Sending ACK from port 1700 to 172.16.1.3/1700

```
IKEv2:(SESSION ID = 52,SA ID = 1):Sending DELETE INFO message for IKEv2 SA [ISPI:
0x5610ADC95CCD6AC7 RSPI: 0x1842E6A7EF7BE81A]
IKEv2:(SESSION ID = 52,SA ID = 1):Building packet for encryption.
Payload contents:
 DELETE NOTIFY(DELETE_REASON)
IKEv2:(SESSION ID = 52,SA ID = 1):Delete all IKE SAs
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Accounting stop request sent successfully
RADIUS(000000D3): Send Accounting-Request to 172.16.1.3:1646 id 1646/97, len 335
RADIUS:  authenticator 9E 05 29 DA EF 2B 44 37 - 89 5A DC F0 0B 71 70 FA
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  61
RADIUS:   Cisco AVpair       [1]   55  "audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN34"
```

```
RADIUS: POD  received from id 9 172.16.1.3:1700, POD Request, len 84
POD: 172.16.1.3 request queued
 ++++++ POD Attribute List ++++++
86D5F0E4 0 00000089 audit-session-id(819) 36 L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN34
POD: 172.16.1.3 Unsupported attribute type 26 for component
POD: 172.16.1.3 user  0.0.0.0i sessid 0x0 key 0x0 DROPPED
POD: Added Reply Message: No Matching Session
POD: Added NACK Error Cause: Invalid Request
POD: Sending NAK from port 1700 to 172.16.1.3/1700
RADIUS:  18  21  4E6F204D61746368696E672053657373696F6E
RADIUS:  101 6   00000194
```

```
Router(config)#aaa server radius ?
  dynamic-author  Local server profile for RFC 3576 support
Router(config)#aaa server radius dynamic-author
Router(config-locsvr-da-radius)#?
RADIUS Application commands:
  client      Specify a RADIUS client
  port        Specify port on which local radius server listens
  server-key  Encryption key shared with the radius clients
```

```
Router(config-locsvr-da-radius)#client ?
  Hostname or A.B.C.D  IP address of RADIUS client
  X:X:X:X::X           IPv6 address of RADIUS client
```

```
Router(config-locsvr-da-radius)#server-key ?
  0      Specifies an UNENCRYPTED key will follow
  6      Specifies an ENCRYPTED key will follow
  7      Specifies HIDDEN key will follow
  WORD   The UNENCRYPTED (cleartext) shared key
```

```
policy-map Input-QoS-policy-1
 class class-default

policy-map Input-QoS-policy-2
 class class-default

policy-map Output-QoS-policy-2
 class class-default

policy-map Output-QoS-policy-1
 class class-default
```

```
interface Virtual-Template1 type tunnel
 ip unnumbered Ethernet0/0
 tunnel protection ipsec profile default
 service-policy input Input-QoS-policy-1
 service-policy output Output-QoS-policy-1
```

```
Router#show crypto session
Crypto session current status
Interface: Virtual-Access1
Profile: default
Session status: UP-ACTIVE
Peer: 172.16.1.1 port 500
  Session ID: 60
  IKEv2 SA: local 172.16.1.2/500 remote 172.16.1.1/500 Active
  IPSEC FLOW: permit 47 host 172.16.1.2 host 172.16.1.1
        Active SAs: 2, origin: crypto-map

Router#show derived-config interface virtual-access 1
interface Virtual-Access1
 ip unnumbered Ethernet0/0
 tunnel source 172.16.1.2
 tunnel destination 172.16.1.1
 tunnel protection ipsec profile default
 no tunnel protection ipsec initiate
 service-policy input Input-QoS-policy-1
 service-policy output Output-QoS-policy-1
end
```

```
Router#show aaa user all | begin router1.domain1
  IPSEC-TUNNEL: Username=router1.domain1
    Session Id=000000F1 Unique Id=000000FB
    Start Sent=1 Stop Only=N
    stop_has_been_sent=N
    Method List=8655B740 : Name = acc_list
    Attribute list:
      86D5F338 0 00000082 formatted-clid(37) 10 172.16.1.1
      86D5F36C 0 0000008A audit-session-id(819) 36
      L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN3C
```

```
RADIUS: COA  received from id 1 172.16.1.3:1700, CoA Request, len 323
COA: 172.16.1.3 request queued
RADIUS:  authenticator 30 46 44 CE 19 6C FE 74 - 1D 64 7A DB 5B 9E 20
0F
RADIUS:  Vendor, Cisco       [26]  64
RADIUS:   Cisco AVpair       [1]   58  "ip:audit-session-id=
L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN3B"
RADIUS:  Vendor, Cisco       [26]  67
RADIUS:   Cisco AVpair       [1]   61  "ip:interface-config=
service-policy input Input-QoS-policy-2"
RADIUS:  Vendor, Cisco       [26]  69
RADIUS:   Cisco AVpair       [1]   63  "ip:interface-config=
service-policy output Output-QoS-policy-2"

 ++++++ CoA Attribute List ++++++
86D5F0E4 0 00000089 audit-session-id(819) 36
L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN3B
86D5F25C 0 00000089 interface-config(222) 39 service-policy input
Input-QoS-policy-2
86D5F290 0 00000089 interface-config(222) 41 service-policy output
Output-QoS-policy-2
```

```
COA: Sending ACK from port 1700 to 172.16.1.3/1700
```

```
VT[Vi1]:Clone Vaccess from AAA (99 bytes)
VT[Vi1]:service-policy input Input-QoS-policy-2
VT[Vi1]:service-policy output Output-QoS-policy-2
VT[Vi1]:end
```

```
Router#show derived-config interface virtual-access 1
interface Virtual-Access1
 ip unnumbered Ethernet0/0
 tunnel source 172.16.1.2
 tunnel destination 172.16.1.1
 tunnel protection ipsec profile default
 no tunnel protection ipsec initiate
 service-policy input Input-QoS-policy-2
 service-policy output Output-QoS-policy-2
end
```

```
Router#show crypto session
Crypto session current status
Interface: Virtual-Access1
Profile: default
Session status: UP-ACTIVE
Peer: 172.16.1.1 port 500
  Session ID: 60
  IKEv2 SA: local 172.16.1.2/500 remote 172.16.1.1/500 Active
  IPSEC FLOW: permit 47 host 172.16.1.2 host 172.16.1.1
        Active SAs: 2, origin: crypto-map

Router#show derived-config interface virtual-access 1
interface Virtual-Access1
 ip unnumbered Ethernet0/0
 tunnel source 172.16.1.2
 tunnel destination 172.16.1.1
 tunnel protection ipsec profile default
 no tunnel protection ipsec initiate
end
```

```
Router#show aaa user all | begin router1.domain1
  IPSEC-TUNNEL: Username=router1.domain1
    Session Id=000000F9 Unique Id=00000103
    Start Sent=1 Stop Only=N
    stop_has_been_sent=N
    Method List=8655B740 : Name = acc_list
    Attribute list:
      86D5EEF8 0 00000082 formatted-clid(37) 10 172.16.1.1
      86D5EF2C 0 0000008A audit-session-id(819) 36
      L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN3D
```

```
RADIUS: COA  received from id 4 172.16.1.3:1700, CoA Request, len 151
COA: 172.16.1.3 request queued
RADIUS:  authenticator AD 54 06 14 D9 22 22 5C - 2E 06 C1 CF 86 56 41 C3
RADIUS:  Vendor, Cisco       [26]  33
RADIUS:   Cisco AVpair       [1]   27  "ip:inacl#3=permit 1.1.1.1"
RADIUS:  Vendor, Cisco       [26]  34
RADIUS:   Cisco AVpair       [1]   28  "ip:outacl#4=permit 1.1.1.3"
RADIUS:  Vendor, Cisco       [26]  64
RADIUS:   Cisco AVpair       [1]   58  "ip:audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN3D"

 ++++++ CoA Attribute List ++++++
86D5F0E4 3 00000081 inacl(144) 14 permit 1.1.1.1
86D5EEC4 4 00000081 outacl(310) 14 permit 1.1.1.3
86D5EEF8 0 00000089 audit-session-id(819) 36
L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZN3D
```

```
COA: Sending ACK from port 1700 to 172.16.1.3/1700
```

```
VT[Vi1]:Clone Vaccess from AAA (86 bytes)
VT[Vi1]:IP access-group Virtual-Access1#2601 in
VT[Vi1]:IP access-group Virtual-Access1#2602 out
VT[Vi1]:end
```

```
Router#show ip interface virtual-access 1 | include access
  Outgoing access list is Virtual-Access1#2602, default is not set
  Inbound  access list is Virtual-Access1#2601, default is not set

Router#show ip access-lists
Standard IP access list Virtual-Access1#2601
    10 permit 1.1.1.1
Standard IP access list Virtual-Access1#2602
    10 permit 1.1.1.3
```

```
<AutoReconnect UserControllable="true">true
  <AutoReconnectBehavior>ReconnectAfterResume</AutoReconnectBehavior>
</AutoReconnect>
```

```
Router(config)#crypto ikev2 profile default
Router(config-ikev2-profile)#authentication local pre-share
Router(config-ikev2-profile)#reconnect ?
  timeout  timeout value for session in reconnect state
  <cr>
Router(config-ikev2-profile)#reconnect
Reconnect can not be configured if either Keyring, PSK  authentication or PSK
authorization enabled in profile.

Router(config-ikev2-profile)#authentication local rsa-sig
Router(config-ikev2-profile)#authentication remote eap
Router(config-ikev2-profile)#reconnect timeout ?
  <600-86400>  reconnect timeout interval in seconds
```

```
ASR-HUB1#show crypto session detail
Crypto session current status

Code: C - IKE Configuration mode, D - Dead Peer Detection
K - Keepalives, N - NAT-traversal, T - cTCP encapsulation
X - IKE Extended Authentication, F - IKE Fragmentation
R - IKE Auto Reconnect

Interface: Virtual-Access1
Profile: ike_profile
Uptime: 00:00:38
Session status: UP-ACTIVE
Peer: 10.104.105.162 port 64696 fvrf: (none) ivrf: (none)
      Phase1_id: eap
      Desc: (none)
  Session ID: 81
  IKEv2 SA: local 10.104.49.17/4500 remote 10.104.105.162/64696 Active
          Capabilities:DNX connid:1 lifetime:23:59:22

Router#show crypto ikev2 sa detail
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         10.104.49.17/4500     10.104.105.162/64696  none/none            READY
      Encr: 3DES, PRF: SHA1, Hash: SHA96, DH Grp:2, Auth sign: RSA, Auth verify: EAP
```

```
Router#show crypto session detail
Crypto session current status

Code: C - IKE Configuration mode, D - Dead Peer Detection
K - Keepalives, N - NAT-traversal, T - cTCP encapsulation
X - IKE Extended Authentication, F - IKE Fragmentation
R - IKE Auto Reconnect

Interface: Virtual-Access1
Profile: ike_profile
Uptime: 00:02:49
Session status: UP-ACTIVE
Peer: 10.104.105.162 port 56028 fvrf: (none) ivrf: (none)
      Phase1_id: eap
      Desc: (none)
  Session ID: 84
  IKEv2 SA: local 10.104.49.17/4500 remote 10.104.105.162/56028 Active
          Capabilities:DNR connid:1 lifetime:23:57:11

Router#show crypto ikev2 sa detail
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         10.104.49.17/4500     10.104.105.162/56028  none/none            READY
      Encr: 3DES, PRF: SHA1, Hash: SHA96, DH Grp:2, Auth sign: RSA, Auth verify: PSK
```

```
Router#show crypto ikev2 stats reconnect
Total incoming reconnect connection:          20
Success reconnect connection:                 20
Failed reconnect connection:                  0
Reconnect capable active session count:       5
Reconnect capable inactive session count:     2
```

```
Router(config-ikev2-profile)#authentication remote ?
  anyconnect-eap  AnyConnect EAP
  eap             Extended Authentication Protocol
  ecdsa-sig       ECDSA Signature
  pre-share       Pre-Shared Key
  rsa-sig         Rivest-Shamir-Adleman Signature
Router(config-ikev2-profile)#authentication remote anyconnect-eap ?
  aggregate  use aggregate auth for anyconnect eap
Router(config-ikev2-profile)#authentication remote anyconnect-eap
aggregate
```

```
Router(config-ikev2-profile)#aaa authentication anyconnect-eap ?
  WORD  AAA list name
```

```
Router(config-ikev2-profile)#aaa authorization group ?
  anyconnect-eap  AAA list to use when IKEv2 remote auth method is
anyconnect
                  eap based

Router(config-ikev2-profile)#aaa authorization user ?
  anyconnect-eap  AAA list to use when IKEv2 remote auth method is
anyconnect
                  eap based

Router(config-ikev2-profile)#aaa accounting ?
  anyconnect-eap  AAA list to use when IKEv2 remote auth method is
AnyConnect
                  EAP
```

```
Router#show crypto ikev2 sa detail
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         192.168.1.1/4500      192.168.1.2/65058     none/none            READY
      Encr: 3DES, PRF: SHA1, Hash: SHA96, DH Grp:2, Auth sign: RSA,
Auth verify: AnyConnect-EAP
      Life/Active Time: 86400/8 sec
      CE id: 1007, Session-id: 6
      Status Description: Negotiation done
      Local spi: 016B20168FB2D051       Remote spi: B4E6576AD4BEAF7A
      Local id: 192.168.1.2
      Remote id: AnyConnect-IKE-ID
      Remote EAP id: user1@cisco.com
```

<IKEIdentity>AnyConnect-IKE-ID</IKEIdentity>

```
Router(config-ikev2-profile)#match identity remote ?
  address  IP Address(es)
  any      match any peer identity
  email    Fully qualified email string [Max. 255 char(s)]
  fqdn     Fully qualified domain name string [Max. 255 char(s)]
  key-id   key-id opaque string

Router(config-ikev2-profile)#match identity remote key-id ?
  WORD  Specify the key-id string

Router(config-ikev2-profile)#match identity remote key-id AnyConnect-IKE-ID
```

```
Router(config-ikev2-profile)#authentication remote anyconnect-eap aggregate ?
  cert-request  use double authentication during anyconnect eap
  <cr>

Router(config-ikev2-profile)#authentication remote anyconnect-eap aggregate
cert-request ?
  <cr>

Router2(config-ikev2-profile)#authentication remote anyconnect-eap aggregate
cert-request
```

```
Router(config-ikev2-profile)#aaa authentication anyconnect-eap ?
  WORD  AAA list name
```

```
FlexClient(config-ikev2-profile)#authentication local eap ?
  gtc        eap method gtc credentials
  md5        eap method md5 credentials
  mschapv2   eap method mschapv2 credentials
  <cr>
```

```
FlexClient(config-ikev2-profile)#authentication local eap gtc ?
  password  EAP password
  username  EAP username

FlexClient(config-ikev2-profile)#authentication local eap md5 ?
  password  EAP password
  username  EAP username

FlexClient(config-ikev2-profile)#authentication local eap mschapv2 ?
  password  EAP password
  username  EAP username
```

```
  crypto eap credentials profile1
Enter the Username for profile profile1: user1
Enter the password for username cisco: password1
```

```
crypto eap credentials profile1
Enter the password for username <IDi>: password1
```

```
crypto ikev2 authorization policy default
 dns 192.168.10.5 192.168.10.6
 split-dns example.net
```

```
ip dns view default
 domain name-server 172.16.10.1
ip dns server
```

```
Received Packet [From 10.10.10.2:500/To 10.10.10.1:500/VRF i0:f0]
Initiator SPI : F09A4EC10E7723C6 - Responder SPI : C980EE3D55880C04 Message id: 1
IKEv2 IKE_AUTH Exchange RESPONSE
Config-type: Config-reply
Attrib type: ipv4-dns, length: 4, data: 192.168.10.5
Attrib type: ipv4-dns, length: 4, data: 192.168.10.6
Attrib type: split-dns, length: 18, data: example.net

FlexVPN(flex1 : 80000009) DNS Primary: 192.168.10.5
FlexVPN(flex1 : 80000009) DNS Secondary: 192.168.10.6
FlexVPN(flex1 : 80000009) Split DNS Name: example.net
```

```
DNS_VIEW: creating view flexvpn-view-flex1

DNS_VIEW: Setting  dns forwarder 192.168.10.5 192.168.10.6 in view
flexvpn-view-flex1

DNS_VIEW: Setting  domain name-server 192.168.10.5 192.168.10.6 in view
flexvpn-view-flex1

DNS_NAMELIST: adding permit 'EXAMPLE.NET' to name-list 1
```

```
FlexClient#show crypto ikev2 sa detailed
      DNS Primary: 192.168.10.5
      DNS Secondary: 192.168.10.6
```

```
FlexClient#show crypto ikev2 client flexvpn flex1 detail
  Split DNS:
   example.net
```

```
FlexClient#show ip dns name-list

ip dns name-list 1

    permit EXAMPLE.NET
```

```
crypto ikev2 authorization policy default
 wins 10.10.210.10
```

```
ip dhcp pool pool1
 import all
 network 10.10.1.0 255.255.255.0
 default-router 10.10.1.1
 dns-server 10.10.1.1


crypto ikev2 client flexvpn flex1
  peer 1 172.16.10.1
  connect manual
  client inside Ethernet0/1
  client connect Tunnel1
```

```
FlexVPNClient#show ip dhcp import

Address Pool Name: pool1

NetBIOS Name Server(s): 10.10.210.10


FlexClient#show crypto ikev2 sa detailed

        WINS Primary: 10.10.210.10
```

```
crypto ikev2 authorization policy default
 wins 10.10.210.10
 def-domain example.net
```

```
ip dhcp pool pool1
 import all
 network 10.10.1.0 255.255.255.0
 default-router 10.10.1
 dns-server 10.10.1.1

crypto ikev2 client flexvpn flex1
  peer 1 203.0.113.224
  connect manual
  client inside Ethernet0/1
  client connect Tunnel1

FlexVPNClient#show ip dhcp import
Address Pool Name: pool1
Domain Name Option: example.net

FlexClient#show crypto ikev2 sa detailed
      Default Domain: example.net
```

peer *sequence* {**ipv4-address** | **ipv6-address** | **fqdn** *fqdn-name* [**dynamic** | ipv6]}
[**track** *track-number* [**up** | **down**]]

```
interface Tunnel10
 ip address negotiated
 tunnel source dynamic

crypto ikev2 client flexvpn flex2
  source 1 Ethernet0/0 track 1
  source 2 Ethernet0/1 track 2
  client connect Tunnel10

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex2
  Current state:ACTIVE
  Peer : 10.0.0.2
  Source : Ethernet0/0
```

```
crypto ikev2 client flexvpn flex2
  peer 1 10.0.0.2 track 1
  peer 2 10.0.0.3 track 2 down
  client connect Tunnel10

interface Tunnel10
 ip address negotiated
 tunnel destination dynamic

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex2
  Current state:ACTIVE
  Peer : 10.0.0.2
```

```
track 10 interface Ethernet0/2 line-protocol
```

```
crypto ikev2 client flexvpn flex1
  peer 1 203.0.113.225
  connect track 10
  client inside Ethernet0/1
  client connect Tunnel1
```

```
%TRACKING-5-STATE: 10 interface Et0/2 line-protocol Down->Up
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel1, changed state to up


%TRACKING-5-STATE: 10 interface Et0/2 line-protocol Up->Down
%FLEXVPN-6-FLEXVPN_CONNECTION_DOWN: FlexVPN(flex1) Client_public_addr =
209.165.201.1 Server_public_addr = 203.0.113.225
```

```
ip sla 1
 icmp-echo 203.0.113.225
ip sla schedule 1 life forever start-time now

track 1 ip sla 1

track 2 ip route 209.165.201.8 255.255.255.255 reachability

track 3 interface Ethernet0/2 line-protocol

track 4 list boolean and
 object 1
 object 2
 object 3 not

FlexClient#show track
Track 1
  IP SLA 1 state
  State is Up
```

```
    1 change, last change 01:30:49
  Latest operation return code: OK
  Latest RTT (millisecs) 1
  Tracked by:
    Track-list 4
Track 2
  IP route 1 209.165.201.8 255.255.255.255 reachability
  Reachability is Up (static)
    1 change, last change 00:27:46
  First-hop interface is Ethernet0/0
  Tracked by:
    Track-list 4
Track 3
  Interface Ethernet0/2 line-protocol
  Line protocol is Down (hw admin-down)
    2 changes, last change 00:00:08
  Tracked by:
    Track-list 4
     0
Track 4
  List boolean and
  Boolean AND is Up
    4 changes, last change 00:00:07
    object 1 Up
    object 2 Up
    object 3 not Down
```

```
crypto ikev2 client flexvpn flex1
  peer 1 172.16.1.1
  backup group 10
  client connect Tunnel1

crypto ikev2 client flexvpn flex2
  peer 1 172.16.2.1
  peer 2 172.16.1.1
  peer 3 172.16.3.1
  backup group 10
  client connect Tunnel2

FlexVPN(flex2 : 8000000B) Current_state: NEGOTIATING
FlexVPN(flex2 : 8000000B) Current_event: EV_TP_ERROR
FlexVPN(flex2 : 8000000B) Error during negotiation
initiating auto reconnect timer
FlexVPN(flex2 : 8000000B) Current_state: NEGOTIATING
FlexVPN(flex2 : 8000000B) Current_event: EV_DISCONNECT
%FLEXVPN-6-FLEXVPN_CONNECTION_DOWN: FlexVPN(flex2) Client_public_addr = 172.16.4.1
Server_public_addr = 172.16.2.1
FlexVPN(flex2 : 0) Connection being terminated with peer 198.51.100.133
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel2, changed state to down
FlexVPN(flex2 : 0) advanced to next peer 172.16.1.1
FlexVPN(flex2 : 0) Current_state: CONNECT_REQUIRED
FlexVPN(flex2 : 0) Current_event: EV_CONNECT
FlexVPN(flex2 : 0) Current_state: CONNECT_REQUIRED
FlexVPN(flex2 : 0) Current_event: EV_SET_PEER
FlexVPN(flex2 : 0) Validating peer 172.16.1.1
FlexVPN(flex2 : 0) Cannot connect to peer 172.16.1.1 since FlexVPN (flex1) is
already connected
FlexVPN(flex2 : 0) Validating peer 172.16.3.1
FlexVPN(flex2 : 0) Ready to connect to peer 172.16.3.1
FlexVPN(flex2 : 0) Current peer set to 172.16.3.1
```

```
ip dhcp pool pool1
 network 172.16.1.0 255.255.255.0

interface Tunnel1
 ip address negotiated
 ip nat outside
 ip virtual-reassembly in
 tunnel source Ethernet0/0
 tunnel destination dynamic
 tunnel protection ipsec profile default

interface Ethernet0/1
 ip address 172.16.1.1 255.255.255.0
 ip nat inside
 ip virtual-reassembly in

access-list 1 permit 172.16.1.0 0.0.0.255

ip nat inside source list 1 interface Tunnel1 overload

FlexClient#show ip nat translations
Pro Inside global      Inside local       Outside local      Outside global
icmp 192.168.1.1:6     172.16.1.2:6       192.168.2.1:6      192.168.2.1:6
tcp 192.168.1.1:22079  172.16.1.2:22079   192.168.2.1:23     192.168.2.1:23
```

```
interface Tunnel 0
 description FlexVPN client tunnel interface
 ip address negotiated
 ip nat outside

interface Ethernet 0/0
 description WAN-interface
 ip nat outside

interface Ethernet 1/0
 description LAN-interface
 ip nat inside

route-map internet-route permit 1
 match interface Ethernet 0/0

route-map flexvpn-route permit 1
 match interface Tunnel 0

ip nat inside source route-map crypto-route interface Tunnel0 overload
ip nat inside source route-map internet-route interface Ethernet0/0 overload
```

```
track 1 stub-object

track 2 list boolean and
 object 1 not

event manager applet intDown
 event syslog pattern "Interface Ethernet0/0, changed state to down"
 action 1 track set 1 state down

event manager applet getIP
 event syslog pattern "Interface Ethernet0/0, changed state to up"
 action 1.0 cli command "enable"
 action 1.1 cli command "show int e0/0 | inc address is [0-9\.]+"
 action 2.0 regexp "address is ([0-9\.]+)" "$_cli_result" match ip
 action 2.1 regexp "(^[0-9]+)\.([0-9]+)" "$ip" match first second
 action 3.0 if $first eq "10"
 action 3.1  syslog msg "This is a RFC 1918 address $ip"
 action 3.2  track set 1 state up
 action 3.3 end
```

```
 action 4.0 if $first eq "192"
 action 4.1  if $second eq "168"
 action 4.2   syslog msg "This is a RFC 1918 address $ip"
 action 4.3   track set 1 state up
 action 4.5  end
 action 4.6 end
 action 5.0 if $first eq "172"
 action 5.1  if $second ge "16"
 action 5.2   if $second le "31"
 action 5.3    syslog msg "This is a RFC 1918 address $ip"
 action 5.4    track set 1 state up
 action 5.5   end
 action 5.6  end
 action 5.7 end


crypto ikev2 client flexvpn flex1
  peer 1 172.16.1.3 track 1
  peer 2 192.168.1.2 track 2
  client connect Tunnel1
```

```
FlexClient#show crypto ikev2 client flexvpn
  Profile : flex_client_profile_1
  Current state:ACTIVE
  Peer : 203.0.113.225
  Source : Ethernet0/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel1
  Assigned IP address: 10.0.0.2

  Profile : flex_client_profile_2
  Current state:CONNECT_REQUIRED
  Backup group: Default
  Tunnel interface : Tunnel2
```

```
FlexClient#show crypto ikev2 sa detailed
        DNS Primary: 192.168.10.5
        DNS Secondary: 192.168.10.6
        WINS Primary: 10.10.210.10
        Default Domain: example.net
```

```
FlexClient#show track
Track 1
  Stub-object
  State is Up
    10 changes, last change 02:23:21, by EEM
  Tracked by:
    Track List 2
    FlexVPN 0
Track 2
  List boolean and
  Boolean AND is Down
    11 changes, last change 02:23:20
    object 1 not Up
  Tracked by:
    FlexVPN 0
```

```
VID  Next payload: VID, reserved: 0x0, length: 23
     43 49 53 43 4F 2D 44 45 4C 45 54 45 2D 52 45 41
     53 4F 4E

 VID  Next payload: NOTIFY, reserved: 0x0, length: 21
     46 4C 45 58 56 50 4E 2D 53 55 50 50 4F 52 54 45
     44
```

```
Router1#show run interface Ethernet0/0
interface Ethernet0/0
 ip address 10.10.10.1 255.255.255.0
 standby 1 ip 10.10.10.100
 standby 1 priority 255
 standby 1 name group1

Router2#show run interface Ethernet0/0
interface Ethernet0/0
 ip address 10.10.10.2 255.255.255.0
 standby 1 ip 10.10.10.100
 standby 1 priority 200
 standby 1 name group1
```

```
Router(config)#crypto ikev2 cluster

Router(config-ikev2-cluster)#
```

```
Router#show crypto ikev2 cluster

Cluster is shutdown
```

```
Router#show crypto ikev2 cluster
Role            : CLB Master
Status          : Up
CLB Slaves      : 2
Cluster IP      : 10.10.10.100
Hold time       : 3000 msec
Overload limit  : 90%
Codes           : '*' Least loaded, '-' Overloaded

Load statistics:
  Gateway           Role   Last-seen  Prio   Load     IKE  Ipsec
  ------------------------------------------------------------
 -10.10.10.2      Master     --       100   90.0%      1      2
 *10.10.10.3       Slave  00:00.728   100   13.2%      0      0
  10.10.10.1       Slave  00:00.766   100   28.2%      1      2
```

```
Router1(config)#crypto ikev2 cluster
Router1(config-ikev2-cluster)#do show crypto ikev2 cluster

Role            : CLB Slave
Status          : Up
Cluster IP      : 10.10.10.100
Hold time       : 3000 msec
Hello-interval  : 1000 msec
Update-interval: 3000 msec

Load statistics:
  Gateway              Last-Ack  Prio   Load      IKE  IPsec
  ------------------------------------------------------------
  10.10.10.3        00:00.725   100   13.2%        0      0


Router1(config-ikev2-cluster)#slave priority 50
Router1(config-ikev2-cluster)#do show crypto ikev2 cluster
```

```
Role            : CLB Slave
Status          : Up
Cluster IP      : 10.10.10.100
Hold time       : 3000 msec
Hello-interval  : 1000 msec
Update-interval: 3000 msec

Load statistics:
  Gateway              Last-Ack   Prio    Load      IKE   IPsec
  ------------------------------------------------------------
  10.10.10.3         00:00.865     50    19.8%        0       0
```

```
crypto ikev2 cluster
 master weight crypto-load 100
 master weight system-load 100
```

```
Router(config)#crypto ikev2 redirect gateway init
 ! (IKEv2 Cluster load-balancer is not enabled)
```

```
crypto ikev2 profile default
 redirect gateway auth | init
```

```
IKEv2 IKE_SA_INIT Exchange RESPONSE
Payload contents:
 NOTIFY(REDIRECT)
IKEv2:(SA ID = 1):Processing IKE_SA_INIT message
IKEv2:(SA ID = 1):Redirect acceptance check with platform
IKEv2:(SA ID = 1):Redirect accepted, clean-up IKEv2 SA, platform will initiate
new request
IKEv2:(SA ID = 1):
IKEv2:(SA ID = 1):Failed SA init exchange
IKEv2:(SA ID = 1):Initial exchange failed
IKEv2:(SA ID = 1):Initial exchange failed
IKEv2:(SA ID = 1):Abort exchange
IKEv2:(SA ID = 1):Deleting SA
```

```
crypto ikev2 client flexvpn flex_client
  peer 1 10.10.10.1
  client connect Tunnel1

interface Tunnel1
 ip address negotiated
 tunnel source Ethernet0/0
 tunnel destination dynamic
 tunnel protection ipsec profile default
```

```
Router#show standby
Ethernet0/0 - Group 1
  State is Listen
  Virtual IP address is 10.10.10.100
  Active virtual MAC address is 0000.0c07.ac01
    Local virtual MAC address is 0000.0c07.ac01 (v1 default)
  Hello time 3 sec, hold time 10 sec
  Preemption disabled
  Active router is 10.10.10.2, priority 255 (expires in 9.120 sec)
  Standby router is 10.10.10.3, priority 100 (expires in 10.816 sec)
  Priority 100 (default 100)
  Group name is "group1" (cfgd)
```

```
NOTIFY(REDIRECTED_FROM)  Next payload: NONE, reserved: 0x0, length: 14
    Security protocol id: Unknown - 0, spi size: 0, type: REDIRECTED_FROM
01 04 0A 0A 0A 64
```

```
interface Rthernet0/1
 ip address 10.10.10.1 255.255.255.0
 standby 1 ip 10.10.10.100
 standby 1 name group1
```

```
crypto ikev2 profile default
 match certificate certmap
 identity local dn
 authentication remote rsa-sig
 authentication local rsa-sig
 pki trustpoint 1
 aaa authorization group cert list default default
 virtual-template 1

crypto ikev2 authorization policy default
 pool pool1
 route set interface

interface Virtual-Template1 type tunnel
 ip unnumbered Loopback1
 tunnel source Ethernet0/1
 tunnel protection ipsec profile default

ip local pool pool1 172.16.1.1 172.16.1.255
```

```
Router1#show crypto ikev2 cluster

Role            : CLB Master
Status          : Up
CLB Slaves      : 2
Cluster IP      : 10.10.10.1
Hold time       : 3000 msec
Overload limit  : 90%
Codes           : '*' Least loaded, '-' Overloaded

Load statistics:
  Gateway           Role   Last-seen  Prio   Load     IKE  IPsec
  ------------------------------------------------------------
  10.10.10.1      Master     --       100   31.7%      1      2
 *10.10.10.3       Slave  00:00.224   100   16.2%      0      0
  10.10.10.2       Slave  00:00.600   100   16.2%      0      0
```

```
IKEv2:Received Packet [From 10.10.20.6:500/To 10.10.10.1:500/VRF i0:f0]
Initiator SPI : 5DF2CD9E0A2E1171 - Responder SPI : 0000000000000000 Message id: 0
IKEv2 IKE_SA_INIT Exchange REQUEST
Payload contents:
 SA KE N VID VID NOTIFY(NAT_DETECTION_SOURCE_IP)
NOTIFY(NAT_DETECTION_DESTINATION_IP) NOTIFY(REDIRECT_SUPPORTED)
```

```
IKEv2:(SESSION ID = 10,SA ID = 1):Redirect check with platform for load-balancing
IKEv2:(SA ID = 1):IKEv2 SA got Redirected to '10.10.10.3'
IKEv2:(SESSION ID = 10,SA ID = 1):Sending Redirect notification

IKEv2:(SESSION ID = 10,SA ID = 1):Sending Packet [To 10.10.20.6:500/From
10.10.10.1:500/VRF i0:f0]
Initiator SPI : 5DF2CD9E0A2E1171 - Responder SPI : 0000000000000000 Message id: 0
IKEv2 IKE_SA_INIT Exchange RESPONSE
Payload contents:
 NOTIFY(REDIRECT)
```

```
crypto ikev2 keyring local_keyring
 peer hub-router
  address 172.16.1.2
  pre-shared-key branch1-hub-key

crypto ikev2 profile default
 match identity remote fqdn hq.example.com
 identity local fqdn branch1.example.com
 authentication local pre-share
 authentication remote pre-share
 keyring local local_keyring
```

```
crypto ikev2 keyring local_keyring
 peer hub-router
  address 172.16.1.2
  pre-shared-key local branch2-to-hub-key
  pre-shared-key remote hub-to-branch2-key

crypto ikev2 profile default
 match identity remote fqdn hq.example.com
 identity local fqdn branch2.example.com
 authentication local pre-share
 authentication remote pre-share
 keyring local local_keyring
```

```
aaa new-model

aaa group server radius radius_group1
 server name radius_server1

aaa authorization network aaa_psk_list group radius_group1

radius server radius_server1
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_server1_key
```

```
crypto ikev2 name-mangler aaa_psk_name_mangler
 fqdn hostname
```

```
crypto ikev2 profile default
 match identity remote fqdn domain example.com
 identity local fqdn hq.example.com
 authentication local pre-share
 authentication remote pre-share
 keyring aaa aaa_psk_list name-mangler aaa_psk_name_mangler
```

```
vsa cisco generic 1 string "ipsec:tunnel-password=branch1-key"
```

```
vsa cisco generic 1 string "ipsec:ikev2-password-local=hub-to-branch2-key"
vsa cisco generic 1 string "ipsec:ikev2-password-remote=branch2-to-hub-key"
```

```
IKEv2-INTERNAL:Fetching shared secret from AAA
IKEv2-INTERNAL:Name-mangler 'aaa_psk_name_mangler' returning mangled-name
'branch1' for name-type 1, name-len 19, name_string branch1.example.com
IKEv2-INTERNAL:Using AAA method list aaa_psk_list for peer branch1
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Password request sent

RADIUS(0000019D): Send Access-Reques t to 172.16.1.3:1645 id 1645/145, len 133
RADIUS:  authenticator 15 BF 00 82 37 B0 1B 60 - C2 55 00 9F 17 23 A9 5B
RADIUS:  User-Name           [1]   9   "branch1"
RADIUS:  User-Password       [2]   18  *
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  62
RADIUS:   Cisco AVpair       [1]   56  "audit-session-id=L2L4AC1G102ZO2L4AC1G101Z
I1F401F4ZM195"
RADIUS:  Service-Type        [6]   6   Outbound                  [5]
RADIUS:  NAS-IP-Address      [4]   6   172.16.1.2
RADIUS(0000019D): Sending a IPv4 Radius Packet
RADIUS(0000019D): Started 5 sec timeout

RADIUS: Received from id 1645/145 172.16.1.3:1645, Access-Accept, len 61
RADIUS:  authenticator E0 55 BA 88 22 3E FC E6 - AF DA 03 B3 22 8C 76 8E
RADIUS:  Vendor, Cisco       [26]  41
RADIUS:   Cisco AVpair       [1]   35  "ipsec:tunnel-password=branch1-key"
RADIUS(0000019D): Received from id 1645/145
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received password response
IKEv2-INTERNAL:Received symmmetric password from radius server
IKEv2:[Crypto Engine -> IKEv2] IKEv2 authentication data generation PASSED
IKEv2:(SESSION ID = 405,SA ID = 1):Verification of peer's authenctication data
PASSED
```

```
IKEv2-INTERNAL:Fetching shared secret from AAA
IKEv2-INTERNAL:Name-mangler 'aaa_psk_name_mangler' returning mangled-name
'branch2' for name-type 1, name-len 19, name_string branch2.example.com
IKEv2-INTERNAL:Using AAA method list aaa_psk_list for peer branch2
IKEv2:(SA ID = 2):[IKEv2 -> AAA] Password request sent

RADIUS(0000019F): Send Access-Request to 172.16.1.3:1645 id 1645/147, len 133
RADIUS:  authenticator E9 CB 55 31 65 06 83 48 - 00 60 15 3E 87 93 3C A9
RADIUS:  User-Name           [1]   9   "branch2"
RADIUS:  User-Password       [2]   18  *
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.4"
RADIUS:  Vendor, Cisco       [26]  62
RADIUS:   Cisco AVpair       [1]   56  "audit-session-
id=L2L4AC1G102ZO2L4AC1G104ZI1F401F4ZM197"
RADIUS:  Service-Type        [6]   6   Outbound                  [5]
RADIUS:  NAS-IP-Address      [4]   6   172.16.1.2
RADIUS(0000019F): Sending a IPv4 Radius Packet
RADIUS(0000019F): Started 5 sec timeout

RADIUS: Received from id 1645/147 172.16.1.3:1645, Access-Accept, len 127
RADIUS:  authenticator F3 20 73 8E A8 B9 9F B8 - F0 02 A4 CA 64 47 66 82
RADIUS:  Vendor, Cisco       [26]  53
RADIUS:   Cisco AVpair       [1]   47  "ipsec:ikev2-password-local=hub-to-
branch2-key"
RADIUS:  Vendor, Cisco       [26]  54
RADIUS:   Cisco AVpair       [1]   48  "ipsec:ikev2-password-remote=branch2-to-
hub-key"
IKEv2:(SA ID = 2):[AAA -> IKEv2] Received password response
IKEv2-INTERNAL:Received local password from radius server
IKEv2-INTERNAL:Received remote password from radius server
IKEv2:[Crypto Engine -> IKEv2] IKEv2 authentication data generation PASSED
IKEv2:(SESSION ID = 407,SA ID = 2):Verification of peer's authenctication data
PASSED
```

```
crypto ikev2 profile default
 match identity remote fqdn hq.example.com
 identity local fqdn branch1.example.com
 authentication local pre-share
 authentication remote pre-share
 keyring local local_keyring

crypto ikev2 client flexvpn flex_client_profile
  peer 1 172.16.1.2
  connect manual
  client connect Tunnel0

interface Tunnel0
 ip address negotiated
 tunnel source Ethernet0/0
 tunnel destination dynamic
 tunnel protection ipsec profile default
```

```
crypto ikev2 profile default
 match identity remote fqdn hq.example.com
 identity local fqdn branch2.example.com
 authentication local pre-share
 authentication remote pre-share
 keyring local local_keyring

crypto ikev2 client flexvpn flex_client_profile
  peer 1 172.16.1.2
  connect manual
  client connect Tunnel0

interface Tunnel0
 ip address negotiated
 tunnel source Ethernet0/0
 tunnel destination dynamic
 tunnel protection ipsec profile default
```

```
aaa new-model

aaa group server radius radius_group1
 server name radius_server1

aaa authorization network user_author_list group radius_group1
aaa authorization network group_author_list local

radius server radius_server1
 address ipv4 172.16.1.3 auth-port 1645 acct-port 1646
 key radius_server1_key
```

```
crypto ikev2 name-mangler user_author_name_mangler
 fqdn hostname

crypto ikev2 name-mangler group_author_name_mangler
 fqdn domain
```

```
crypto ikev2 authorization policy example.com
 dns 192.168.10.10
```

```
crypto ikev2 profile default
 match identity remote fqdn domain example.com
 identity local fqdn hq.example.com
 authentication local pre-share
 authentication remote pre-share
 keyring aaa aaa_psk_list name-mangler aaa_psk_name_mangler
 aaa authorization group psk list group_author_list name-mangler group_author_
name_mangler
 aaa authorization user psk list user_author_list name-mangler user_author_name_
mangler
 virtual-template 1
```

```
framed address 10.0.0.1 255.255.255.0
```

```
framed address 10.0.0.2 255.255.255.0
```

```
IKEv2:Using mlist group_author_list and username example.com for group author
request
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorisation request sent
IKEv2-INTERNAL:IKEv2 local AAA author request for 'example.com'
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response
IKEv2-INTERNAL:Received group author attributes:
ipv4-dns: 192.168.10.10, route-accept any tag:1 distance:1,
```

```
IKEv2:Using mlist user_author_list and username branch1 for user author request
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorisation request sent
RADIUS(0000001D): Send Access-Request to 172.16.1.3:1645 id 1645/10, len 131
RADIUS:  authenticator 76 2E F4 8B 41 A0 49 05 - E4 29 56 6A D3 6D 23 97
RADIUS:  User-Name           [1]   9   "branch1"
RADIUS:  User-Password       [2]   18  *
RADIUS:  Calling-Station-Id  [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco       [26]  60
RADIUS:   Cisco AVpair       [1]   54  "audit-session-
id=L2L4AC1G102ZO2L4AC1G101ZI1F401F4ZO6"
RADIUS:  Service-Type        [6]   6   Outbound                  [5]
RADIUS:  NAS-IP-Address      [4]   6   172.16.1.2
RADIUS(0000001D): Sending a IPv4 Radius Packet
RADIUS(0000001D): Started 5 sec timeout
RADIUS: Received from id 1645/10 172.16.1.3:1645, Access-Accept, len 73
RADIUS:  authenticator D5 CE AC 26 AA 67 89 A6 - 35 BC 09 FC 08 6B E9 F6
RADIUS:  Framed-IP-Address   [8]   6   10.0.0.1
RADIUS:  Framed-IP-Netmask   [9]   6   255.255.255.0
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response
IKEv2-INTERNAL:Received user author attributes:
ipv4-addr: 10.0.0.1, ipv4-netmask: 255.255.255.0,
```

```
IKEv2:(SESSION ID = 6,SA ID = 1):Config-type: Config-reply
IKEv2:(SESSION ID = 6,SA ID = 1):Attrib type: ipv4-addr, length: 4, data: 10.0.0.1
IKEv2:(SESSION ID = 6,SA ID = 1):Attrib type: ipv4-netmask, length: 4, data:
255.255.255.0
IKEv2:(SESSION ID = 6,SA ID = 1):Attrib type: ipv4-dns, length: 4, data:
192.168.10.10
```

```
FlexClient#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA
Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         172.16.1.1/500        172.16.1.2/500        none/none            READY
      Encr: AES-CBC, keysize: 256, PRF: SHA512, Hash: SHA512, DH Grp:5, Auth sign:
PSK, Auth verify: PSK
      Life/Active Time: 86400/29 sec
      CE id: 1004, Session-id: 4
      Status Description: Negotiation done
      Local spi: 7BFDBB152E002029       Remote spi: E8700C8FDC94AA9A
      Local id: branch1.example.com
      Remote id: hq.example.com
      Pushed IP address: 10.0.0.1
      DNS Primary: 192.168.10.10
```

```
IKEv2:Using mlist group_author_list and username example.com for group author
request
IKEv2:(SA ID = 2):[IKEv2 -> AAA] Authorisation request sent
IKEv2-INTERNAL:IKEv2 local AAA author request for 'example.com'
IKEv2:(SA ID = 2):[AAA -> IKEv2] Received AAA authorisation response
IKEv2-INTERNAL:Received group author attributes:
ipv4-dns: 192.168.10.10, route-accept any tag:1 distance:1,
```

```
IKEv2:Using mlist user_author_list and username branch2 for user author request
IKEv2:(SA ID = 2):[IKEv2 -> AAA] Authorization request sent
RADIUS(00000021): Send Access-Request to 172.16.1.3:1645 id 1645/12, len 131
RADIUS:  authenticator CB BD 72 FD 0E 06 B4 DA - 29 4F 1D 59 56 52 25 0E
RADIUS:  User-Name            [1]   9   "branch2"
RADIUS:  User-Password        [2]   18  *
RADIUS:  Calling-Station-Id   [31]  12  "172.16.1.4"
RADIUS:  Vendor, Cisco        [26]  60
RADIUS:   Cisco AVpair        [1]   54  "audit-session-
id=L2L4AC1G102ZO2L4AC1G104ZI1F401F4ZO7"
RADIUS:  Service-Type         [6]   6   Outbound                  [5]
RADIUS:  NAS-IP-Address       [4]   6   172.16.1.2
RADIUS(00000021): Sending a IPv4 Radius Packet
RADIUS(00000021): Started 5 sec timeout
RADIUS: Received from id 1645/12 172.16.1.3:1645, Access-Accept, len 139
RADIUS:  authenticator 8D 16 2E 75 1C E8 9D 67 - CA E1 F1 E4 33 47 79 71
RADIUS:  Framed-IP-Address    [8]   6   10.0.0.2
RADIUS:  Framed-IP-Netmask    [9]   6   255.255.255.0
RADIUS(00000021): Received from id 1645/12
IKEv2:(SA ID = 2):[AAA -> IKEv2] Received AAA authorization response
IKEv2-INTERNAL:Received user author attributes:
ipv4-addr: 10.0.0.2, ipv4-netmask: 255.255.255.0,
```

```
IKEv2:(SESSION ID = 7,SA ID = 2):Config-type: Config-reply
IKEv2:(SESSION ID = 7,SA ID = 2):Attrib type: ipv4-addr, length: 4, data: 10.0.0.2
IKEv2:(SESSION ID = 7,SA ID = 2):Attrib type: ipv4-netmask, length: 4, data:
255.255.255.0
IKEv2:(SESSION ID = 7,SA ID = 2):Attrib type: ipv4-dns, length: 4, data:
192.168.10.10
```

```
FlexClient2#show crypto ikev2 sa detailed
IPv4 Crypto IKEv2  SA
Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         172.16.1.4/500        172.16.1.2/500        none/none            READY
      Encr: AES-CBC, keysize: 256, PRF: SHA512, Hash: SHA512, DH Grp:5, Auth sign:
PSK, Auth verify: PSK
      Life/Active Time: 86400/7 sec
      CE id: 1003, Session-id: 3
      Status Description: Negotiation done
      Local spi: 93E45A454B87A9D5       Remote spi: 841BBA759B822DBF
      Local id: branch2.example.com
      Remote id: hq.example.com
      Pushed IP address: 10.0.0.2
      DNS Primary: 192.168.10.10
```

```
aaa new-model

aaa authorization network group_author_list local
```

```
crypto ikev2 name-mangler group_author_name_mangler
 fqdn domain
```

```
crypto ikev2 authorization policy example.com
 route set interface Ethernet1/0
 route set interface
```

```
crypto ikev2 profile default
 match identity remote fqdn hq.example.com
 identity local fqdn branch1.example.com
 authentication local pre-share key cisco
 authentication remote pre-share key cisco
 aaa authorization group psk list group_author_list name-mangler group_author_
name_mangler
```

```
interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 ipv6 address FEC0::10:1/120
 tunnel source Ethernet0/0
 tunnel destination 172.16.1.2
 tunnel protection ipsec profile default
```

```
interface Ethernet0/0
 description IPv4 WAN interface
 ip address 172.16.1.1 255.255.255.0
```

```
interface Ethernet1/0
 description LAN interface
 ip address 192.168.0.1 255.255.255.0
 ipv6 address FEC0::1:1/120
```

```
aaa new-model

aaa authorization network group_author_list local
```

```
crypto ikev2 name-mangler group_author_name_mangler
 fqdn domain
```

```
crypto ikev2 authorization policy example.com
 route set interface
 route set remote ipv4 192.168.1.0 255.255.255.0
 route set remote ipv6 FEC0::2:1/120
```

```
crypto ikev2 profile default
 match identity remote fqdn hq.example.com
 identity local fqdn branch2.example.com
 authentication local pre-share key cisco
 authentication remote pre-share key cisco
 aaa authorization group psk list group_author_list name-mangler group_author_
name_mangler
```

```
interface Tunnel0
 ip address 10.0.0.2 255.255.255.0
 ipv6 address FEC0::10:2/120
 tunnel source Ethernet0/0
 tunnel mode gre ipv6
 tunnel destination 2001::1:2
 tunnel protection ipsec profile default
```

```
interface Ethernet0/0
 description IPv6 WAN interface
 no ip address
 ipv6 address 2001::1:1/120
```

```
interface Ethernet1/0
 description LAN interface
 ip address 192.168.1.1 255.255.255.0
 ipv6 address FEC0::2:1/120
```

```
aaa new-model

aaa authorization network group_author_list local
```

```
crypto ikev2 name-mangler group_author_name_mangler
 fqdn domain
```

```
crypto ikev2 authorization policy example.com
 route set interface
 route set access-list v4-subnets
 route set access-list ipv6 v6-subnets

ip access-list standard v4-subnets
 permit 192.168.2.0 0.0.0.255

ipv6 access-list v6-subnets
 permit ipv6 FEC0::3:0/120 any
```

```
crypto ikev2 profile default
 match identity remote fqdn domain example.com
 identity local fqdn hq.example.com
 authentication local pre-share key cisco
 authentication remote pre-share key cisco
 aaa authorization group psk list group_author_list name-mangler group_author_
name_mangler
 virtual-template 1 mode auto
```

```
interface Loopback1
 ip address 10.0.0.3 255.255.255.0
 ipv6 address FEC0::10:3/120

interface Virtual-Template1 type tunnel
 ip unnumbered Loopback1
 ipv6 unnumbered Loopback1
 tunnel protection ipsec profile default
```

```
interface Ethernet0/0
 description IPv4 and IPv6 WAN interface
 ip address 172.16.1.2 255.255.255.0
 ipv6 address 2001::1:2/120
```

```
interface Ethernet1/0
 description LAN interface
 ip address 192.168.2.1 255.255.255.0
 ipv6 address FEC0::3:1/120
```

```
FlexSpoke1#show crypto session

FlexSpoke1#show ip route static

FlexSpoke1#
```

```
FlexSpoke1#show crypto session
Crypto session current status
Interface: Tunnel0
Profile: default
Session status: UP-ACTIVE
Peer: 172.16.1.2 port 500
  Session ID: 1
  IKEv2 SA: local 172.16.1.1/500 remote 172.16.1.2/500 Active
  IPSEC FLOW: permit 47 host 172.16.1.1 host 172.16.1.2
        Active SAs: 2, origin: crypto-map
```

```
FlexSpoke1#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA
Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         172.16.1.1/500        172.16.1.2/500        none/none            READY
      Remote subnets:
      10.0.0.3 255.255.255.255
      192.168.2.0 255.255.255.0
      IPv6 Remote subnets:
      FEC0::10:3/128
      FEC0::3:0/120

FlexSpoke1#show ip route static
      10.0.0.0/8 is variably subnetted, 3 subnets, 2 masks
S        10.0.0.3/32 is directly connected, Tunnel0
S     192.168.2.0/24 is directly connected, Tunnel0

FlexSpoke1#show ipv6 route static
S   FEC0::3:0/120 [1/0], tag 1
     via Tunnel0, directly connected
S   FEC0::10:3/128 [1/0], tag 1
     via Tunnel0, directly connected
```

```
FlexSpoke1#clear crypto sa counters
FlexSpoke1#show crypto session detail | begin IPSEC
  IPSEC FLOW: permit 47 host 172.16.1.1 host 172.16.1.2
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 0 drop 0 life (KB/Sec) 4248161/2860
        Outbound: #pkts enc'ed 0 drop 0 life (KB/Sec) 4248160/2860

FlexSpoke1#ping ipv6 FEC0::3:1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to FEC0::3:1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexSpoke1#show crypto session detail | begin IPSEC
  IPSEC FLOW: permit 47 host 172.16.1.1 host 172.16.1.2
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 5 drop 0 life (KB/Sec) 4248160/2841
        Outbound: #pkts enc'ed 5 drop 0 life (KB/Sec) 4248159/2841

FlexSpoke1#ping 192.168.2.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.2.1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexSpoke1#show crypto session detail | begin IPSEC
  IPSEC FLOW: permit 47 host 172.16.1.1 host 172.16.1.2
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 10 drop 0 life (KB/Sec) 4248158/2805
        Outbound: #pkts enc'ed 10 drop 0 life (KB/Sec) 4248158/2805
```

```
FlexSpoke2#show crypto session

FlexSpoke2#show ip route static

FlexSpoke2#
```

```
FlexSpoke2#show crypto session
Crypto session current status
Interface: Tunnel0
Profile: default
Session status: UP-ACTIVE
Peer: 2001::1:2 port 500
  Session ID: 1
  IKEv2 SA: local 2001::1:1/500
          remote 2001::1:2/500 Active
  IPSEC FLOW: permit 47 host 2001::1:1 host 2001::1:2
        Active SAs: 2, origin: crypto-map
```

```
FlexSpoke2#show crypto ikev2 sa detailed
IPv6 Crypto IKEv2  SA
Tunnel-id    fvrf/ivrf                Status
1           none/none                READY
Local  2001::1:1/500
Remote  2001::1:2/500

      Remote subnets:
      10.0.0.3 255.255.255.255
      192.168.2.0 255.255.255.0
      IPv6 Remote subnets:
      FEC0::10:3/128
      FEC0::3:0/120

FlexSpoke2#show ip route static
      10.0.0.0/8 is variably subnetted, 3 subnets, 2 mas
S        10.0.0.3/32 is directly connected, Tunnel0
S     192.168.2.0/24 is directly connected, Tunnel0

FlexSpoke2#show ipv6 route static
S   FEC0::3:0/120 [1/0], tag 1
     via Tunnel0, directly connected
S   FEC0::10:3/128 [1/0], tag 1
     via Tunnel0, directly connected
```

```
FlexSpoke2#clear crypto sa counters
FlexSpoke2#show crypto session detail | begin IPSEC
  IPSEC FLOW: permit 47 host 2001::1:1 host 2001::1:2
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 0 drop 0 life (KB/Sec) 4346787/3407
        Outbound: #pkts enc'ed 0 drop 0 life (KB/Sec) 4346787/3407

FlexSpoke2#ping ipv6 FEC0::3:1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to FEC0::3:1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexSpoke2#show crypto session detail | begin IPSEC
  IPSEC FLOW: permit 47 host 2001::1:1 host 2001::1:2
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 5 drop 0 life (KB/Sec) 4346787/3387
        Outbound: #pkts enc'ed 5 drop 0 life (KB/Sec) 4346786/3387
```

```
FlexSpoke2#ping 192.168.2.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.2.1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexSpoke2#show crypto session detail | begin IPSEC
  IPSEC FLOW: permit 47 host 2001::1:1 host 2001::1:2
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 10 drop 0 life (KB/Sec) 4346785/3361
        Outbound: #pkts enc'ed 10 drop 0 life (KB/Sec) 4346785/3361
```

```
FlexHub#show crypto session
Crypto session current status
Interface: Virtual-Access2
Profile: default
Session status: UP-ACTIVE
Peer: 172.16.1.1 port 500
  Session ID: 5
  IKEv2 SA: local 172.16.1.2/500 remote 172.16.1.1/500 Active
  IPSEC FLOW: permit 47 host 172.16.1.2 host 172.16.1.1
        Active SAs: 2, origin: crypto-map
FlexHub#show derived-config interface virtual-access 2
Derived configuration : 220 bytes

interface Virtual-Access2
 ip unnumbered Loopback1
 ipv6 unnumbered Loopback1
 tunnel source 172.16.1.2
 tunnel destination 172.16.1.1
 tunnel protection ipsec profile default
 no tunnel protection ipsec initiate
end

FlexHub#show interfaces virtual-access 2 | include protocol/transport
  Tunnel protocol/transport GRE/IP
```

```
FlexHub#show crypto session
Crypto session current status
Interface: Virtual-Access1
Profile: default
Session status: UP-ACTIVE
Peer: 2001::1:1 port 500
  Session ID: 6
  IKEv2 SA: local 2001::1:2/500
          remote 2001::1:1/500 Active
  IPSEC FLOW: permit 47 host 2001::1:2 host 2001::1:1
        Active SAs: 2, origin: crypto-map

FlexHub#show derived-config interface virtual-access 1
Derived configuration : 240 bytes

interface Virtual-Access1
 ip unnumbered Loopback1
 ipv6 unnumbered Loopback1
 tunnel source 2001::1:2
 tunnel mode gre ipv6
 tunnel destination 2001::1:1
 tunnel protection ipsec profile default
 no tunnel protection ipsec initiate
end

FlexHub#show interfaces virtual-access 1 | include protocol/transport
  Tunnel protocol/transport GRE/IPv6
```

```
FlexHub#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA
Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         172.16.1.2/500        172.16.1.1/500        none/none            READY
      Remote subnets:
      10.0.0.1 255.255.255.255
      192.168.0.0 255.255.255.0
      IPv6 Remote subnets:
      FEC0::10:1/128
      FEC0::1:0/120

 IPv6 Crypto IKEv2  SA
Tunnel-id    fvrf/ivrf              Status
2          none/none              READY
Local  2001::1:2/500
Remote  2001::1:1/500
            Remote subnets:
      10.0.0.2 255.255.255.255
      192.168.1.0 255.255.255.0
      IPv6 Remote subnets:
      FEC0::10:2/128
      FEC0::2:1/120

FlexHub#show ip route static
      10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
```

```
S        10.0.0.1/32 is directly connected, Virtual-Access2
S        10.0.0.2/32 is directly connected, Virtual-Access1
S     192.168.0.0/24 is directly connected, Virtual-Access2
S     192.168.1.0/24 is directly connected, Virtual-Access1

FlexHub#show ipv6 route static
S   FEC0::1:0/120 [1/0], tag 1
     via Virtual-Access2, directly connected
S   FEC0::2:0/120 [1/0], tag 1
     via Virtual-Access1, directly connected
S   FEC0::10:1/128 [1/0], tag 1
     via Virtual-Access2, directly connected
S   FEC0::10:2/128 [1/0], tag 1
     via Virtual-Access1, directly connected
```

```
FlexHub#ping 192.168.0.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.0.1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexHub#ping ipv6 FEC0::1:1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to FEC0::1:1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexHub#show crypto session interface virtual-access 2 detail | begin IPSEC
  IPSEC FLOW: permit 47 host 172.16.1.2 host 172.16.1.1
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 10 drop 0 life (KB/Sec) 4200943/2000
        Outbound: #pkts enc'ed 10 drop 0 life (KB/Sec) 4200944/2000
```

```
FlexHub#ping 192.168.1.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexHub#ping ipv6 FEC0::2:1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to FEC0::2:1, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexHub#show crypto session interface virtual-access 1 detail | begin IPSEC
  IPSEC FLOW: permit 47 host 2001::1:2 host 2001::1:1
        Active SAs: 2, origin: crypto-map
        Inbound:  #pkts dec'ed 10 drop 0 life (KB/Sec) 4339994/2178
        Outbound: #pkts enc'ed 10 drop 0 life (KB/Sec) 4339996/2178
```

```
aaa new-model

aaa authorization network group_author_list local

crypto ikev2 name-mangler group_author_name_mangler
 fqdn domain
```

```
crypto ikev2 authorization policy default
 route set interface

crypto ikev2 profile default
 match identity remote fqdn hq.example.com
 identity local fqdn branch1.example.com
 authentication local pre-share key cisco
 authentication remote pre-share key cisco
 aaa authorization group psk list group_author_list default

crypto ikev2 client flexvpn flex_client_profile
  peer 1 172.16.1.2
  connect manual
  client connect Tunnel0
```

```
interface Loopback0
 description simulates LAN subnet
 ip address 192.168.0.1 255.255.255.0
 ip nat inside
 ip virtual-reassembly in
```

```
interface Tunnel0
 ip address negotiated
 ip nat outside
 ip virtual-reassembly in
 tunnel source Ethernet0/0
 tunnel destination dynamic
 tunnel protection ipsec profile default

interface Ethernet0/0
 description WAN interface
 ip address 172.16.1.1 255.255.255.0
```

```
ip nat inside source route-map flex-route-map interface Tunnel0 overload

route-map flex-route-map permit 1
 match interface Tunnel0
```

```
FlexClient#show crypto ikev2 sa detail
 IPv4 Crypto IKEv2  SA
Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         172.16.1.1/500        172.16.1.2/500        none/none            READY
      Pushed IP address: 10.0.0.1
      DNS Primary: 192.168.10.10
      Remote subnets:
      10.0.0.10 255.255.255.255
      192.168.1.0 255.255.255.0

FlexClient#show ip route static
      4.0.0.0/24 is subnetted, 1 subnets
S        4.4.4.0 is directly connected, Tunnel0
      10.0.0.0/8 is variably subnetted, 3 subnets, 2 masks
S        10.0.0.10/32 is directly connected, Tunnel0
S     192.168.1.0/24 is directly connected, Tunnel0
```

```
FlexClient#ping 192.168.1.1 source 192.168.0.1 repeat 1
Type escape sequence to abort.
Sending 1, 100-byte ICMP Echos to 192.168.1.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.0.1

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

FlexClient#
IP: s=192.168.0.1 (local), d=192.168.1.1, len 100, local feature, NAT(2), rtype 0,
forus FALSE, sendself FALSE, mtu 0, fwdchk FALSE
IP: s=192.168.0.1 (local), d=192.168.1.1 (Tunnel0), len 100, sending
```

```
IP: s=10.0.0.1 (local), d=192.168.1.1 (Tunnel0), len 100, output feature, Post-
routing NAT Outside(26), rtype 1, forus FALSE, sendself FALSE, mtu 0, fwdchk FALSE
```

```
IP: s=10.0.0.1 (local), d=192.168.1.1 (Tunnel0), len 100, post-encap feature,
IPSEC Post-encap output classification(16), rtype 1, forus FALSE, sendself FALSE,
mtu 0, fwdchk FALSE

IP: s=10.0.0.1 (local), d=192.168.1.1 (Tunnel0), len 100, sending full packet

IP: s=172.16.1.1 (local), d=172.16.1.2 (Ethernet0/0), g=172.16.1.2, len 168,
forward

IP: s=172.16.1.1 (local), d=172.16.1.2 (Ethernet0/0), len 168, sending full packet
```

```
FlexClient#show ip nat translations
Pro Inside global      Inside local       Outside local      Outside global
icmp 10.0.0.1:15       192.168.0.1:15     192.168.1.1:15     192.168.1.1:15

FlexClient#show ip nat statistics
Total active translations: 1 (0 static, 1 dynamic; 1 extended)
Peak translations: 4, occurred 01:13:02 ago
Outside interfaces:
  Tunnel0
Inside interfaces:
  Loopback0
Hits: 65  Misses: 0
CEF Translated packets: 30, CEF Punted packets: 0
Expired translations: 13
Dynamic mappings:
-- Inside Source
[Id: 1] route-map flex-route-map interface Tunnel0 refcount 1
```

```
track 1 interface Serial0/0 line-protocol

track 2 interface Serial1/0 line-protocol
```

```
crypto ikev2 client flexvpn flex-client
  peer 1 172.16.3.1
  source 1 Serial0/0 track 1
  source 2 Serial1/0 track 2
  client connect Tunnel0
```

```
interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 ipv6 address FEC0::10:1/120
 tunnel source dynamic
 tunnel destination dynamic
 tunnel protection ipsec profile default
```

```
interface Serial0/0
 description MPLS
 ip address 172.16.1.1 255.255.255.0
 keepalive 2
 serial restart-delay 0

interface Serial1/0
 description Internet
 ip address 172.16.2.1 255.255.255.0
 keepalive 2
 serial restart-delay 0
```

```
ip route 172.16.3.1 255.255.255.255 Serial0/0
ip route 172.16.3.1 255.255.255.255 Serial1/0
```

```
FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:ACTIVE
  Peer : 172.16.3.1
  Source : Serial0/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel0
```

```
%TRACK-6-STATE: 1 interface Se0/0 line-protocol Up -> Down
* %FLEXVPN-6-FLEXVPN_CONNECTION_DOWN: FlexVPN(flex-client) Client_public_addr =
172.16.1.1 Server_public_addr = 172.16.3.1
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to down
%CRYPTO-6-ISAKMP_ON_OFF: ISAKMP is OFF
%CRYPTO-6-ISAKMP_ON_OFF: ISAKMP is ON
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0, changed state to down
```

```
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up
%FLEXVPN-6-FLEXVPN_CONNECTION_UP: FlexVPN(flex-client) Client_public_addr =
172.16.2.1 Server_public_addr = 172.16.3.1

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:ACTIVE
  Peer : 172.16.3.1
  Source : Serial1/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel0
```

```
TRACK-6-STATE: 1 interface Se0/0 line-protocol Down -> Up
%FLEXVPN-6-FLEXVPN_CONNECTION_DOWN: FlexVPN(flex-client) Client_public_addr =
172.16.2.1 Server_public_addr = 172.16.3.1
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to down
%CRYPTO-6-ISAKMP_ON_OFF: ISAKMP is OFF
%CRYPTO-6-ISAKMP_ON_OFF: ISAKMP is ON

%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0, changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up
%FLEXVPN-6-FLEXVPN_CONNECTION_UP: FlexVPN(flex-client) Client_public_addr =
172.16.1.1 Server_public_addr = 172.16.3.1

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:ACTIVE
  Peer : 172.16.3.1
  Source : Serial0/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel0
```

```
track 1 ip sla 1 reachability


ip sla 1
 icmp-echo 172.16.2.1
 frequency 5
ip sla schedule 1 start-time now
```

```
crypto ikev2 client flexvpn flex-client
   peer 1 172.16.2.1 track 1
   peer 2 172.16.3.1
   peer reactivate
   client connect Tunnel0
```

```
interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination dynamic
 tunnel protection ipsec profile default
```

```
interface Ethernet0/0
 description Internet
 ip address 172.16.1.1 255.255.255.0
```

```
FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:ACTIVE
  Peer : 172.16.2.1
  Source : Ethernet0/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel0
```

```
%TRACK-6-STATE: 1 ip sla 1 reachability Up -> Down
%FLEXVPN-6-FLEXVPN_CONNECTION_DOWN: FlexVPN(flex-client) Client_public_addr =
172.16.1.1 Server_public_addr = 172.16.2.1
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to down
```

```
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up
%FLEXVPN-6-FLEXVPN_CONNECTION_UP: FlexVPN(flex-client) Client_public_addr =
172.16.1.1 Server_public_addr = 172.16.3.1

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:ACTIVE
  Peer : 172.16.3.1
  Source : Ethernet0/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel0
```

```
%TRACK-6-STATE: 1 ip sla 1 reachability Down -> Up
%FLEXVPN-6-FLEXVPN_CONNECTION_DOWN: FlexVPN(flex-client) Client_public_addr =
172.16.1.1 Server_public_addr = 172.16.3.1
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to down


%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up
%FLEXVPN-6-FLEXVPN_CONNECTION_UP: FlexVPN(flex-client) Client_public_addr =
172.16.1.1 Server_public_addr = 172.16.2.1

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:ACTIVE
  Peer : 172.16.2.1
  Source : Ethernet0/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel0
```

```
track 1 interface Serial1/0 line-protocol
```

```
crypto ikev2 client flexvpn flex-client
  peer 1 172.16.1.2
  connect track 1 down
  client connect Tunnel0

interface Tunnel0
 ip address 10.0.0.1 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination dynamic
 tunnel protection ipsec profile default
```

```
interface Ethernet0/0
 description Internet
 ip address 172.16.1.1 255.255.255.0


interface Serial1/0
 description MPLS
 ip address 172.16.2.1 255.255.255.0
 keepalive 2
 serial restart-delay 0
```

```
FlexClient#show interfaces Serial 1/0 | include line protocol
Serial1/0 is up, line protocol is up

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:CONNECT_REQUIRED
  Backup group: Default
  Tunnel interface : Tunnel0
```

```
%TRACK-6-STATE: 1 interface Se1/0 line-protocol Up -> Down
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up
%FLEXVPN-6-FLEXVPN_CONNECTION_UP: FlexVPN(flex-client) Client_public_addr =
172.16.1.1 Server_public_addr = 172.16.1.2

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:ACTIVE
  Peer : 172.16.1.2
  Source : Ethernet0/0
  ivrf : IP DEFAULT
  fvrf : IP DEFAULT
  Backup group: Default
  Tunnel interface : Tunnel0
```

```
%TRACK-6-STATE: 1 interface Se1/0 line-protocol Down -> Up
%FLEXVPN-6-FLEXVPN_CONNECTION_DOWN:FlexVPN(flex-client) Client_public_addr =
172.16.1.1 Server_public_addr = 172.16.1.2

FlexClient#show crypto ikev2 client flexvpn
  Profile : flex-client
  Current state:CONNECT_REQUIRED
  Backup group: Default
  Tunnel interface : Tunnel0
```

```
snmp-server context SNMP-WAN
```

```
vrf definition wan
    address-family ipv4
     snmp context   SNMP-WAN
    exit-address-family
```

```
snmp-server group wan-grp   v3 auth context SNMP-WAN
```

```
snmp-server user wan-user wan-grp v3 auth sha cisco123
```

```
snmp-server host 10.10.10.1 vrf mgmt version 3 auth  wan-user
```

```
service timestamps debug datetime localtime show-timezone
service timestamps log datetime localtime show-timezone
clock timezone GMT 0
```

```
%LINEPROTO-5-UPDOWN: Line protocol on Interface Ethernet0/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface Ethernet0/0, changed state to down
```

```
snmp-server enable traps snmp linkup linkdown
```

```
SNMP: V1 Trap, ent snmpTraps, addr 10.10.10.1, gentrap 2, spectrap 0
 ifIndex.1 = 1
 ifDescr.1 = Ethernet0/0
 ifType.1 = 6
 lifEntry.20.1 = administratively down
```

```
%DUAL-5-NBRCHANGE: IP-EIGRP 109: Neighbor 10.1.1.1 (GigabitEthernet1/2) is
down: peer restarted
%DUAL-5-NBRCHANGE: IP-EIGRP 109: Neighbor 10.1.1.1 (GigabitEthernet1/2) is
down: interface down
%DUAL-5-NBRCHANGE: IP-EIGRP 109: Neighbor 10.1.1.1 (GigabitEthernet1/2) is up:
new adjacency

%BGP-5-ADJCHANGE: neighbor 10.1.1.1 Up
%BGP-5-ADJCHANGE: neighbor 10.1.1.1 Down BGP Notification sent
%BGP-5-ADJCHANGE: neighbor 10.1.1.1 Up
```

```
snmp-server enable traps eigrp
SNMP can be used to alert on
SNMP: Queuing packet to 10.10.10.1
SNMP: V1 Trap, ent ciscoEigrpMIB, addr 10.10.10.1, gentrap 6, spectrap 3
 cEigrpPeerAddr.0.1.0 = 10.10.50.8
 cEigrpPeerAddrType.0.1.0 = 1
```

```
snmp-server enable traps ike tunnel start
```

```
SNMPv2-MIB::snmpTrapOID.0 = OID: CISCO-IPSEC-FLOW-MONITOR-MIB::cikeTunnelStart
CISCO-IPSEC-FLOW-MONITOR-MIB::cikePeerLocalAddr.9."hostname=router1.cisco.com".9
"hostname=router2.cisco.com".9 = Hex-STRING: 0A 0A 0A 02
CISCO-IPSEC-FLOW-MONITOR-MIB::cikePeerRemoteAddr.9."hostname=router1.cisco.com".9
"hostname=router2.cisco.com"9 = Hex-STRING: 0A 0A 0A 01

CISCO-IPSEC-FLOW-MONITOR-MIB::cikeTunLifeTime.9 = INTEGER: 86400 seconds
```

```
snmp-server enable traps ike tunnel stop
```

```
SNMPv2-MIB::snmpTrapOID.0 = OID: CISCO-IPSEC-FLOW-MONITOR-MIB::cikeTunnelStop
CISCO-IPSEC-FLOW-MONITOR-MIB::cikePeerLocalAddr.9."hostname=router1.cisco.com".9
"hostname=router2.cisco.com".9 = Hex-STRING: 0A 0A 0A 02
CISCO-IPSEC-FLOW-MONITOR-MIB::cikePeerRemoteAddr.9."hostname=router1.cisco.com".9
"hostname=router2.cisco.com"9 = Hex-STRING: 0A 0A 0A 01


CISCO-IPSEC-FLOW-MONITOR-MIB::cikeTunActiveTime.3 = INTEGER: 11
CISCO-IPSEC-FLOW-MONITOR-MIB::cikeTunHistTermReason.3 = INTEGER: other(1)
```

```
Router#show crypto mib ike flowmib ?
  failure   Shows ike mib failure table
  global    Shows ike mib global statistics
  history   Shows ike mib history table
  peer      Shows ike mib peer info
  tunnel    Shows ike mib tunnel info
```

```
crypto mib ipsec flow history tunnel size <2-200>
```

```
Router#show snmp mib | i cike
cikeGlobalActiveTunnels
cikeGlobalPreviousTunnels
cikeGlobalInOctets
cikeGlobalInPkts
cikeGlobalInDropPkts
cikeGlobalInNotifys
cikeGlobalInP2Exchgs
cikeGlobalInP2ExchgInvalids
cikeGlobalInP2ExchgRejects
cikeGlobalInP2SaDelRequests
```

```
Router(config)#crypto logging ikev2
```

```
Router(config)#crypto logging session
```

```
%CRYPTO-5-IKEV2_SESSION_STATUS: Crypto tunnel v2 is UP.  Peer 10.10.10.1:500
Id: hostname=Router.cisco.com
```

```
%CRYPTO-5-IKEV2_SESSION_STATUS: Crypto tunnel v2 is DOWN.  Peer 10.10.10.1:500
Id: hostname=Router1.cisco.com
```

```
%IKEV2-3-NEG_ABORT: Negotiation aborted due to ERROR: Auth exchange failed


%IKEV2-3-NEG_ABORT: Negotiation aborted due to ERROR: Received no proposal chosen
notify
```

```
AAA/ACCT/EVENT/(00000011): CALL START
AAA/ACCT/EVENT/(00000011): IPSEC TNL UP
AAA/ACCT/IPSEC-TUNNEL(00000011): Queueing record is START

IKEv2:(SA ID = 1):[IKEv2 -> AAA] Accounting start request sent successfully
RADIUS(00000011): Send Accounting-Request to 172.16.1.3:1646 id 1646/9, len 256
RADIUS:  authenticator 6A 7A 8F EA 4E 10 E9 CE - 8C C9 EC A8 C4 70 C7 9F
RADIUS:  Acct-Session-Id      [44]  10  "00000007"
RADIUS:  Calling-Station-Id   [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco        [26]  60
RADIUS:   Cisco AVpair        [1]   54  "audit-session-id=L2L4AC1G102ZO2L4AC1G101Z
I1F401F4ZO8"
RADIUS:  Vendor, Cisco        [26]  40
RADIUS:   Cisco AVpair        [1]   34  "isakmp-phase1-id=Router1.Domain1"
RADIUS:  Vendor, Cisco        [26]  37
RADIUS:   Cisco AVpair        [1]   31  "isakmp-initator-ip=172.16.1.1"
RADIUS:  User-Name            [1]   17  "Router1.Domain1"
RADIUS:  Vendor, Cisco        [26]  36
RADIUS:   Cisco AVpair        [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Authentic       [45]  6   Local                       [2]
RADIUS:  Acct-Status-Type     [40]  6   Start                       [1]
RADIUS:  NAS-IP-Address       [4]   6   172.16.1.2
RADIUS:  Acct-Delay-Time      [41]  6   0
```

```
AAA/ACCT/EVENT/(00000011): IPSEC TNL DOWN
IKEv2:in_octets 13020, out_octets 17640
IKEv2:in_packets 105, out_packets 105
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Accounting stop request sent successfully
AAA/ACCT/EVENT/(00000011): CALL STOP
AAA/ACCT/IPSEC-TUNNEL(00000011): STOP

RADIUS(00000011): Send Accounting-Request to 172.16.1.3:1646 id 1646/10, len 324
RADIUS:  authenticator 6E 18 B4 4F E0 B5 C0 09 - 9F B9 FC 01 5A D4 69 6F
RADIUS:  Acct-Session-Id      [44]  10  "00000007"
RADIUS:  Calling-Station-Id   [31]  12  "172.16.1.1"
RADIUS:  Vendor, Cisco        [26]  60
RADIUS:   Cisco AVpair        [1]   54  "audit-session-id=L2L4AC1G102ZO2L4AC1G101Z
I1F401F4ZO8"
RADIUS:  Vendor, Cisco        [26]  40
RADIUS:   Cisco AVpair        [1]   34  "isakmp-phase1-id=Router1.Domain1"
RADIUS:  Vendor, Cisco        [26]  37
RADIUS:   Cisco AVpair        [1]   31  "isakmp-initator-ip=172.16.1.1"
RADIUS:  User-Name            [1]   17  "Router1.Domain1"
RADIUS:  Acct-Authentic       [45]  6   Local                       [2]
RADIUS:  Vendor, Cisco        [26]  36
RADIUS:   Cisco AVpair        [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Session-Time    [46]  6   34
RADIUS:  Acct-Input-Octets    [42]  6   13020
RADIUS:  Acct-Output-Octets   [43]  6   17640
RADIUS:  Acct-Input-Packets   [47]  6   105
RADIUS:  Acct-Output-Packets  [48]  6   105
RADIUS:  Acct-Terminate-Cause[49]  6   none                        [0]
RADIUS:  Vendor, Cisco        [26]  32
RADIUS:   Cisco AVpair        [1]   26  "disc-cause-ext=No Reason"
RADIUS:  Acct-Status-Type     [40]  6   Stop                        [2]
RADIUS:  NAS-IP-Address       [4]   6   172.16.1.2
RADIUS:  Acct-Delay-Time      [41]  6   0
```

```
aaa new-model

aaa group server radius RA-AUTHC-SG-1
 server-private 172.16.100.100 auth-port 1812 key Cisco123

aaa accounting network RA-ACCC-LIST-1 start-stop group RA-AUTHC-SG-1

crypto ikev2 profile ALL-SPOKES
 match certificate certmap
 identity local dn
 authentication remote rsa-sig
 authentication local rsa-sig
 pki trustpoint MYCERT
 aaa accounting cert RA-ACCC-LIST-1
 virtual-template 1
```

```
RADIUS(00000016): Send Accounting-Request to 172.16.100.100:1646 id 1646/17,
len 284
RADIUS:  authenticator 1D C9 AB 28 97 97 98 D7 - 01 F1 35 3D 79 F1 53 16
RADIUS:  Acct-Session-Id      [44]  10  "0000000A"
RADIUS:  Calling-Station-Id   [31]  11  "209.1.1.2"
RADIUS:  Vendor, Cisco        [26]  62
RADIUS:   Cisco AVpair        [1]   56  "audit-session-id=L2L4C8010102ZO2L4D101010
2ZI1F401F4ZOE"
RADIUS:  Vendor, Cisco        [26]  54
RADIUS:   Cisco AVpair        [1]   48  "isakmp-phase1-id=hostname=Spoke-1.cisco
.com"
RADIUS:  Vendor, Cisco        [26]  36
RADIUS:   Cisco AVpair        [1]   30  "isakmp-initator-ip=209.1.1.2"
RADIUS:  User-Name            [1]   31  "hostname=Spoke-1.cisco.com"
RADIUS:  Vendor, Cisco        [26]  36
RADIUS:   Cisco AVpair        [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Authentic       [45]  6   Local                       [2]
RADIUS:  Acct-Status-Type     [40]  6   Start                       [1]
Hub-1(config-ikev2-profile)#
RADIUS:  NAS-IP-Address       [4]   6   172.16.0.1
RADIUS:  Acct-Delay-Time      [41]  6   0
RADIUS(00000016): Sending a IPv4 Radius Packet
RADIUS(00000016): Started 5 sec timeout
```

```
RADIUS(00000016): Send Accounting-Request to 172.16.100.100:1646 id 1646/26,
len 352
RADIUS:  authenticator D2 06 49 A8 79 F8 77 04 - E3 58 B9 4F D5 66 5E E8
RADIUS:  Acct-Session-Id      [44]  10  "0000000A"
RADIUS:  Calling-Station-Id   [31]  11  "209.1.1.2"
RADIUS:  Vendor, Cisco        [26]  62
RADIUS:   Cisco AVpair        [1]   56  "audit-session-id=L2L4C8010102ZO2L4D101010
2ZI1F401F4ZOE"
RADIUS:  Vendor, Cisco        [26]  54
RADIUS:   Cisco AVpair        [1]   48  "isakmp-phase1-id=hostname=Spoke-1.cisco
.com"
RADIUS:  Vendor, Cisco        [26]  36
RADIUS:   Cisco AVpair        [1]   30  "isakmp-initator-ip=209.1.1.2"
RADIUS:  User-Name            [1]   31  "hostname=Spoke-1.cisco.com"
RADIUS:  Acct-Authentic       [45]  6   Local                       [2]
RADIUS:  Vendor, Cisco        [26]  36
RADIUS:   Cisco AVpair        [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Session-Time    [46]  6   39
RADIUS:  Acct-Input-Octets    [42]  6   15222
RADIUS:  Acct-Output-Octets   [43]  6   15641
RADIUS:  Acct-Input-Packets   [47]  6   22
RADIUS:  Acct-Output-Packets  [48]  6   17
RADIUS:  Acct-Terminate-Cause[49]  6   none                        [0]
RADIUS:  Vendor, Cisco        [26]  32
RADIUS:   Cisco AVpair        [1]   26  "disc-cause-ext=No Reason"
RADIUS:  Acct-Status-Type     [40]  6   Stop                        [2]
RADIUS:  NAS-IP-Address       [4]   6   172.16.0.1
RADIUS:  Acct-Delay-Time      [41]  6   0
RADIUS(00000016): Sending a IPv4 Radius Packet
```

```
aaa new-model

aaa group server radius RA-AUTHC-SG-1
 server-private 172.16.100.100 auth-port 1812 key Cisco123

aaa authentication login RA-AUTHC-LIST-1 group RA-AUTHC-SG-1
aaa authorization network RA-AUTHZ-LIST-1 local
aaa accounting network RA-ACCC-LIST-1 start-stop group RA-AUTHC-SG-1

crypto ikev2 profile RA-CLIENT
 match identity remote address 0.0.0.0
 identity local dn
 authentication remote eap query-identity
 authentication local rsa-sig
 pki trustpoint MYCERT
 aaa authentication eap RA-AUTHC-LIST-1
 aaa authorization group eap list RA-AUTHZ-LIST-1 RA-LOCAL-POLICY-1
 aaa accounting eap RA-ACCC-LIST-1
 virtual-template 10
```

```
RADIUS/ENCODE: Best Local IP-Address 172.16.0.1 for Radius-Server 172.16.100.100
RADIUS(00000014): Send Accounting-Request to 172.16.100.100:1646 id 1646/1, len
256
RADIUS:  authenticator 50 3B BC 65 EB 53 6A 91 - 44 13 F0 FE 6D 06 01 2F
RADIUS:  Calling-Station-Id  [31]  14  "209.1.10.100"
RADIUS:  Vendor, Cisco       [26]  63
RADIUS:   Cisco AVpair       [1]   57  "audit-session-id=L2L4C8010102ZO2L4D1010A6
4ZH11941194ZO6"
RADIUS:  Vendor, Cisco       [26]  37
RADIUS:   Cisco AVpair       [1]   31  "isakmp-phase1-id=209.1.10.100"
RADIUS:  Vendor, Cisco       [26]  39
RADIUS:   Cisco AVpair       [1]   33  "isakmp-initator-ip=209.1.10.100"
RADIUS:  Framed-IP-Address   [8]   6   172.16.101.10
RADIUS:  Acct-Session-Id     [44]  10  "00000006"
RADIUS:  User-Name           [1]   7   "user1"
RADIUS:  Vendor, Cisco       [26]  36
Hub-1#
RADIUS:   Cisco AVpair       [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Authentic      [45]  6   Local                      [2]
RADIUS:  Acct-Status-Type    [40]  6   Start                      [1]
RADIUS:  NAS-IP-Address      [4]   6   172.16.0.1
RADIUS:  Acct-Delay-Time     [41]  6   0
```

```
RADIUS(00000014): Send Accounting-Request to 172.16.100.100:1646 id 1646/6, len 324
RADIUS:  authenticator B5 59 09 6E F0 A5 8D F0 - 37 1B 64 53 C5 AC 00 91
RADIUS:  Calling-Station-Id  [31]  14  "209.1.10.100"
RADIUS:  Vendor, Cisco       [26]  63
RADIUS:   Cisco AVpair       [1]   57  "audit-session-id=L2L4C8010102ZO2L4D1010A6
4ZH11941194ZO6"
RADIUS:  Vendor, Cisco       [26]  37
RADIUS:   Cisco AVpair       [1]   31  "isakmp-phase1-id=209.1.10.100"
RADIUS:  Vendor, Cisco       [26]  39
RADIUS:   Cisco AVpair       [1]   33  "isakmp-initator-ip=209.1.10.100"
RADIUS:  Framed-IP-Address   [8]   6   172.16.101.10
RADIUS:  Acct-Session-Id     [44]  10  "00000006"
RADIUS:  User-Name           [1]   7   "user1"
RADIUS:  Acct-Authentic      [45]  6   Local                    [2]
RADIUS:  Vendor, Cisco       [26]  36
RADIUS:   Cisco AVpair       [1]   30  "connect-progress=No Progress"
RADIUS:  Acct-Session-Time   [46]  6   51
RADIUS:  Acct-Input-Octets   [42]  6   7577
RADIUS:  Acct-Output-Octets  [43]  6   3064
RADIUS:  Acct-Input-Packets  [47]  6   91
RADIUS:  Acct-Output-Packets [48]  6   23
RADIUS:  Acct-Terminate-Cause[49]  6   none                     [0]
RADIUS:  Vendor, Cisco       [26]  32
RADIUS:   Cisco AVpair       [1]   26  "disc-cause-ext=No Reason"
RADIUS:  Acct-Status-Type    [40]  6   Stop                     [2]
RADIUS:  NAS-IP-Address      [4]   6   172.16.0.1
RADIUS:  Acct-Delay-Time     [41]  6   0
```

```
IKEv2:Using mlist default and username service1 for group author request
AAA/BIND(0000004E): Bind i/f
AAA/AUTHOR (0x4E): Pick method list 'default'
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorisation request sent
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response
```

```
snmp-server enable traps ipsec tunnel start
```

```
snmp-server enable traps ipsec tunnel start
```

```
SNMPv2-MIB::snmpTrapOID.0 = OID: CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecTunnelStop

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecTunActiveTime.13 = INTEGER: 301032

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecTunHistActiveTime.13 = INTEGER: 301032

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecTunHistLifeTime.13 = INTEGER: 3600 Seconds

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecTunHistLifeSize.13 = INTEGER: 4608000 KBytes

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecTunHistTermReason.13 = INTEGER: peerLost (5)

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistLocalType.13 = INTEGER:
singleIpAddr(1)

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistRemoteType.13 = INTEGER:
singleIpAddr(1)

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistLocalAddr1.13 = Hex-STRING: 0A 0A
0A 01

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistLocalAddr2.13 = Hex-STRING: 0A 0A
0A 01

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistRemoteAddr1.13 = Hex-STRING: 0A 0A
0A 02

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistRemoteAddr2.13 = Hex-STRING: 0A 0A
0A 02

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistLocalProtocol.13 = INTEGER: 47

 CISCO-IPSEC-FLOW-MONITOR-MIB::cipSecEndPtHistLocalPort.13 = INTEGER: 0
```

```
Router#show snmp mib | i cipSec
cipSecMibLevel
cipSecGlobalActiveTunnels
cipSecGlobalPreviousTunnels
cipSecGlobalInOctets
cipSecGlobalHcInOctets
cipSecGlobalInOctWraps
cipSecGlobalInDecompOctets
cipSecGlobalHcInDecompOctets
cipSecGlobalInDecompOctWraps
```

```
snmp-server enable traps snmp linkdown linkup
```

```
ifIndex.24 = 24
  ifDescr.24 = Virtual-Access1
  ifType.24 = 131
  lifEntry.20.24 = Tunnel Up

ifIndex.24 = 24
  ifDescr.24 = Virtual-Access1
  ifType.24 = 131
  lifEntry.20.24 = Tunnel Down
```

```
Router#debug ip routing static detail

%CRYPTO-5-IKEV2_SESSION_STATUS: Crypto tunnel v2 is UP.  Peer 10.10.60.6:500
f_vrf:  OUTSIDE i_vrf:  OUTSIDE   Id: hostname=Router.cisco.com
%LINEPROTO-5-UPDOWN: Line protocol on Interface Virtual-Access3, changed state to
up

IP-ST(default): updating same distance on 192.168.2.37/32
IP-ST(default):  192.168.2.37/32 [1], Virtual-Access3 Path =, add succeed, active
state
```

```
Router#show ip route 192.168.2.36
Routing entry for 192.168.2.36/32
  Known via "static", distance 1, metric 0 (connected)
  Tag 1
  Routing Descriptor Blocks:
  * directly connected, via Virtual-Access1, permanent
      Route metric is 0, traffic share count is 1
      Route tag 1
```

```
flow record v4_r1
match ipv4 tos
match ipv4 protocol
match ipv4 source address
match ipv4 destination address
match transport source-port
match transport destination-port
collect counter bytes long
collect counter packets long
```

```
interface Virtual-Template2 type tunnel
 ip unnumbered Loopback2
 ip flow monitor f-monitor-1 input
 ip flow monitor f-monitor-1 output
```

```
Router#show derived-config interface virtual-access 2
Building configuration...

Derived configuration : 421 bytes

interface Virtual-Access2
 ip unnumbered Loopback2
 ip flow monitor f-monitor-1 input
 ip flow monitor f-monitor-1 output
```

```
Router#show flow monitor f-monitor-1 cache
  Cache type:                                 Normal
  Cache size:                                   4096
  Current entries:                                 1
  High Watermark:                                  3

  Flows added:                                     8
  Flows aged:                                      7
    - Active timeout      (  1800 secs)            0
    - Inactive timeout    (    15 secs)            7
    - Event aged                                   0
    - Watermark aged                               0
    - Emergency aged                               0

IPV4 SOURCE ADDRESS:       192.168.2.4
IPV4 DESTINATION ADDRESS:  192.168.2.1
TRNS SOURCE PORT:          38658
```

```
TRNS DESTINATION PORT:      23
IP TOS:                     0xC0
IP PROTOCOL:                6
counter bytes long:         1545
counter packets long:       37

Router#show flow exporter statistics
Flow Exporter exp-1:
  Packet send statistics (last cleared 00:20:18 ago):
    Successfully sent:          4                       (438 bytes)

  Client send statistics:
    Client: Flow Monitor f-monitor-1
      Records added:            3
        - sent:                 3
      Bytes added:              90
        - sent:                 90
```

```
Router#show monitor event-trace crypto ikev2 ?
  error       Show IKEv2 Errors
  event       Show IKEv2 Events
  exception   Show IKEv2 Exceptions
```

```
Router#show monitor event-trace crypto ikev2 error ?
  all         Show all the traces in current buffer
  back        Show trace from this far back in the past
  clock       Show trace from a specific clock time/date
  from-boot   Show trace from this many seconds after booting
  latest      Show latest trace events since last display
  parameters  Parameters of the trace
```

```
Router#show monitor event-trace crypto ipsec ?
  error       Show IPSEC Error
  event       Show IPSEC Events
  exception   Show IPSEC Exception
```

```
Router#show monitor event-trace crypto ipsec error ?
  all          Show all the traces in current buffer
  back         Show trace from this far back in the past
  clock        Show trace from a specific clock time/date
  from-boot    Show trace from this many seconds after booting
  latest       Show latest trace events since last display
  parameters   Parameters of the trace
```

```
Router#show monitor event-trace crypto ikev2 error all
SA ID:1 SESSION ID:1 Remote: 10.10.10.1/500 Local:
                      10.10.60.6/500
SA ID:1 SESSION ID:1 Remote: 10.10.10.1/500 Local:
                      10.10.60.6/500  : Auth exchange failed
SA ID:1 SESSION ID:1 Remote: 10.10.10.1/500 Local:
                      10.10.60.6/500  Negotiation aborted due to ERROR: Auth
                      exchange failed
ID:0 SESSION ID:0    Optional profile description not
                      updated in PSH
```

```
Router(config)#monitor event-trace crypto ikev2 error ?
  dump-file   Set name of trace dump file
  size        Set size of trace
  stacktrace  Trace call stack at tracepoints; clear the trace buffer first
  <cr>
```

```
Router#debug crypto ikev2

IKEv2 default debugging is on
```

```
Router#debug crypto ikev2 ?
  client    Client
  cluster   IKEv2 Cluster load-balancer debugging
  error     IKEv2 Error debugging
  internal  IKEv2 Internal debugging
  packet    IKEv2 Packet debugging
```

```
IKEv2-PAK:(SESSION ID = 1,SA ID = 1):Next payload: SA, version: 2.0 Exchange type:
IKE_SA_INIT, flags: INITIATOR Message id: 0, length: 464
Payload contents:
 SA  Next payload: KE, reserved: 0x0, length: 48
  last proposal: 0x0, reserved: 0x0, length: 44
  Proposal: 1, Protocol id: IKE, SPI size: 0, #trans: 4    last transform: 0x3,
reserved: 0x0: length: 12
    type: 1, reserved: 0x0, id: AES-CBC
    last transform: 0x3, reserved: 0x0: length: 8
    type: 2, reserved: 0x0, id: SHA256
    last transform: 0x3, reserved: 0x0: length: 8
    type: 3, reserved: 0x0, id: SHA256
    last transform: 0x0, reserved: 0x0: length: 8
    type: 4, reserved: 0x0, id: DH_GROUP_2048_MODP/Group 14
 KE  Next payload: N, reserved: 0x0, length: 264
    DH group: 14, Reserved: 0x0
 N  Next payload: VID, reserved: 0x0, length: 24
 VID  Next payload: VID, reserved: 0x0, length: 23
 VID  Next payload: NOTIFY, reserved: 0x0, length: 21
 NOTIFY(NAT_DETECTION_SOURCE_IP)  Next payload: NOTIFY, reserved: 0x0, length: 28
    Security protocol id: IKE, spi size: 0, type: NAT_DETECTION_SOURCE_IP
 NOTIFY(NAT_DETECTION_DESTINATION_IP)  Next payload: NONE, reserved: 0x0, length: 28
    Security protocol id: IKE, spi size: 0, type: NAT_DETECTION_DESTINATION_IP
```

```
IKEv2-PAK:(SESSION ID = 1,SA ID = 1):Next payload: SA, version: 2.0 Exchange type:
IKE_SA_INIT, flags: INITIATOR Message id: 0, length: 464
Payload contents:
 SA  Next payload: KE, reserved: 0x0, length: 48
  last proposal: 0x0, reserved: 0x0, length: 44
  Proposal: 1, Protocol id: IKE, SPI size: 0, #trans: 4    last transform: 0x3,
reserved: 0x0: length: 12
    type: 1, reserved: 0x0, id: AES-CBC
    last transform: 0x3, reserved: 0x0: length: 8
    type: 2, reserved: 0x0, id: SHA256
    last transform: 0x3, reserved: 0x0: length: 8
    type: 3, reserved: 0x0, id: SHA256
    last transform: 0x0, reserved: 0x0: length: 8
    type: 4, reserved: 0x0, id: DH_GROUP_2048_MODP/Group 14
 KE  Next payload: N, reserved: 0x0, length: 264
    DH group: 14, Reserved: 0x0

     45 B7 59 62 46 9A 5A 61 7A 93 FC 63 DC 5E 87 AC
     50 AA DB 2F 11 39 19 F5 7F E4 6C C3 57 39 60 E6
     C9 ED E4 A1 4E A1 87 87 12 79 D7 DD 68 83 06 2B
     32 25 C2 D6 01 AD F3 4F 40 23 2C 02 AA 16 6A 25
     45 B5 D2 61 B9 DD 4C 72 95 27 08 13 E4 FE E1 32
     19 F3 5A A3 0E EF 59 2A CC C4 A2 97 E4 04 73 21
     82 BB 71 4B 85 CB BE 33 55 89 61 D8 28 4A B7 CB
     32 18 AB A0 CB 5D AB C1 4D 82 3A 4B 51 41 B4 A4
```

```
   5B 56 C2 73 B3 B0 26 67 57 05 55 9D C2 BE 46 BC
   9F 1A EA 6C CA D3 07 65 02 D4 EC 32 A9 B3 94 ED
   F6 D6 EE 54 FB 74 30 F4 E3 DE E2 57 0B 50 4E 88
   E9 3C 8A 35 44 6F 34 42 DA B6 8D 49 B7 0A 8A 68
   90 A9 C7 39 C5 35 98 42 0F B4 91 F3 31 E7 42 D9
   7D 20 43 AE B5 C4 B0 34 2C 16 15 1E 9A 68 F4 F6
   C3 11 BE 46 AC C7 25 44 23 35 AF 6E EB 6B 74 CC
   14 BF F2 40 7D E6 48 B7 9A 3C C1 C4 0B A3 26 BD

N  Next payload: VID, reserved: 0x0, length: 24

   E4 77 64 DC 98 E7 3E 85 7C 41 CA A1 21 50 2F 0F
   6C AC AE A4

VID  Next payload: VID, reserved: 0x0, length: 23

   43 49 53 43 4F 2D 44 45 4C 45 54 45 2D 52 45 41
   53 4F 4E

VID  Next payload: NOTIFY, reserved: 0x0, length: 21

   40 55 3D 1F 3C D1 36 43 82 0D 0F C5 15 ED 49 9E

NOTIFY(NAT_DETECTION_SOURCE_IP)  Next payload: NOTIFY, reserved: 0x0, length: 28
  Security protocol id: IKE, spi size: 0, type: NAT_DETECTION_SOURCE_IP

   0D BE 1C 55 0D 3F A6 61 EE C8 F3 1D 8C 6B 46 9D
   E7 09 23 75
```

NOTIFY(NAT_DETECTION_DESTINATION_IP) Next payload: NONE, reserved: 0x0, length: 28
Security protocol id: IKE, spi size: 0, type: NAT_DETECTION_DESTINATION_IP

53 CD 98 45 F3 B9 1F 39 8E EB 2C CB C2 AA CF DE
64 C2 85 19

IKEv2 Tx packet(on wire):

F4 9D AD A2 60 CB 37 3F 00 00 00 00 00 00 00 00
21 20 22 08 00 00 00 00 00 00 01 D0 22 00 00 30
00 00 00 2C 01 01 00 04 03 00 00 0C 01 00 00 0C
80 0E 00 80 03 00 00 08 02 00 00 05 03 00 00 08
03 00 00 0C 00 00 00 08 04 00 00 0E 28 00 01 08
00 0E 00 00 45 B7 59 62 46 9A 5A 61 7A 93 FC 63
DC 5E 87 AC 50 AA DB 2F 11 39 19 F5 7F E4 6C C3
57 39 60 E6 C9 ED E4 A1 4E A1 87 87 12 79 D7 DD
68 83 06 2B 32 25 C2 D6 01 AD F3 4F 40 23 2C 02
AA 16 6A 25 45 B5 D2 61 B9 DD 4C 72 95 27 08 13
E4 FE E1 32 19 F3 5A A3 0E EF 59 2A CC C4 A2 97
E4 04 73 21 82 BB 71 4B 85 CB BE 33 55 89 61 D8
28 4A B7 CB 32 18 AB A0 CB 5D AB C1 4D 82 3A 4B
51 41 B4 A4 5B 56 C2 73 B3 B0 26 67 57 05 55 9D
C2 BE 46 BC 9F 1A EA 6C CA D3 07 65 02 D4 EC 32
A9 B3 94 ED F6 D6 EE 54 FB 74 30 F4 E3 DE E2 57
0B 50 4E 88 E9 3C 8A 35 44 6F 34 42 DA B6 8D 49
B7 0A 8A 68 90 A9 C7 39 C5 35 98 42 0F B4 91 F3
31 E7 42 D9 7D 20 43 AE B5 C4 B0 34 2C 16 15 1E
9A 68 F4 F6 C3 11 BE 46 AC C7 25 44 23 35 AF 6E
EB 6B 74 CC 14 BF F2 40 7D E6 48 B7 9A 3C C1 C4
0B A3 26 BD 2B 00 00 18 E4 77 64 DC 98 E7 3E 85
7C 41 CA A1 21 50 2F 0F 6C AC AE A4 2B 00 00 17
43 49 53 43 4F 2D 44 45 4C 45 54 45 2D 52 45 41
53 4F 4E 29 00 00 15 46 4C 45 58 56 50 4E 2D 53
55 50 50 4F 52 54 45 44 29 00 00 1C 01 00 40 04
0D BE 1C 55 0D 3F A6 61 EE C8 F3 1D 8C 6B 46 9D
E7 09 23 75 00 00 00 1C 01 00 40 05 53 CD 98 45
F3 B9 1F 39 8E EB 2C CB C2 AA CF DE 64 C2 85 19

```
Router#debug crypto ipsec ?
  client       Client Debug
  error        IPSEC errors
  ha           IPSEC High Availability
  hw-request   IPSEC hw-request
  message      IPSEC message
  metadata     CTS metadata
  states       IPSEC states
```

```
KMI: (Session ID: 1) IPSEC key engine sending message KEY_ENG_REQUEST_SAS to
Crypto IKEv2.

KMI: (Session ID: 1) Crypto IKEv2 received message KEY_ENG_REQUEST_SAS from IPSEC
key engine.

KMI: (Session ID: 1) Crypto IKEv2 sending message KEY_MGR_VALIDATE_IPSEC_PROPOSALS
to IPSEC key engine.

KMI: (Session ID: 1) IPSEC key engine received message KEY_MGR_VALIDATE_IPSEC_
PROPOSALS from Crypto IKEv2.

KMI: (Session ID: 1) Crypto IKEv2 sending message KEY_MGR_CREATE_IPSEC_SAS to
IPSEC key engine.

KMI: (Session ID: 1) IPSEC key engine received message KEY_MGR_CREATE_IPSEC_SAS
from Crypto IKEv2.

KMI: IPSEC key engine sending message KEY_ENG_NOTIFY_INCR_COUNT to Crypto IKEv2.
```

```
KMI: (Session ID: 3) Crypto IKEv2 sending message KEY_MGR_IKMP_READY to IPSEC key
engine.

KMI: (Session ID: 3) IPSEC key engine received message KEY_MGR_IKMP_READY from
Crypto IKEv2.


KMI: IPSEC key engine sending message KEY_ENG_IPSEC_READY to Crypto IKEv2.

KMI: Crypto IKEv2 received message KEY_ENG_IPSEC_READY from IPSEC key engine.

KMI: (Session ID: 3) Crypto IKEv2 sending message KEY_MGR_VALIDATE_IPSEC_PROPOSALS
to IPSEC key engine.

KMI: (Session ID: 3) IPSEC key engine received message
KEY_MGR_VALIDATE_IPSEC_PROPOSALS from Crypto IKEv2.

KMI: (Session ID: 3) Crypto IKEv2 sending message KEY_MGR_CREATE_IPSEC_SAS to
IPSEC key engine.

KMI: (Session ID: 3) IPSEC key engine received message KEY_MGR_CREATE_IPSEC_SAS
from Crypto IKEv2.

KMI: IPSEC key engine sending message KEY_ENG_NOTIFY_INCR_COUNT to Crypto IKEv2.

KMI: Crypto IKEv2 received message KEY_ENG_NOTIFY_INCR_COUNT from IPSEC
key engine.
```

```
Router#debug crypto condition ?
  unmatched  Output debugs even if no context available
```

```
Router#monitor capture buffer buf1 size 2048 max-size 1600 linear
```

```
Router(config)#ip access-list buf-filter
Router(config-ext-nacl)#permit udp any any eq 500

Router#monitor capture buffer buf1 filter access-list buf-filter
```

```
Router#monitor capture point ip cef point1 GigabitEthernet0/0 in
```

```
Router#monitor capture point associate point1 buf1
```

```
Router#monitor capture point start point1

%BUFCAP-6-ENABLE: Capture Point point1 enabled
```

```
Router#monitor capture point stop point1

%BUFCAP-6-DISABLE: Capture Point point1 disabled
```

```
Router#show monitor capture buffer buf1 dump
UTC May 21 2016 : IPv4 LES CEF    : Gi0/0 None

2D2EBFA0:                                     A44C110E                  $L..
2D2EBFB0: B4A0C89C 1DFEBF20 080045C0 00420058  4 H..~? ..E@.B.X
2D2EBFC0: 0000FE11 610DAC10 0101AC10 016401F4  ..~.a.,...,..d.t
<removed>
```

```
monitor cap buffer buf1 export flash:/buf1.pcap
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):IKE Proposal: 1, SPI size: 0 (initial
negotiation),

Num. transforms: 15

   AES-CBC   AES-CBC   AES-CBC   SHA512   SHA384   SHA256   SHA1   MD5   SHA512
SHA384   SHA256   SHA96   MD596   DH_GROUP_1536_MODP/Group 5
DH_GROUP_1024_MODP/Group 2


 IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 10.10.10.1:500/From
10.10.20.3:500/VRF i0:f0]

Initiator SPI : 502A6C10FFD6F8BC - Responder SPI : 0000000000000000 Message id: 0


IKEv2 IKE_SA_INIT Exchange REQUEST

Payload contents:

 SA KE N VID VID NOTIFY(NAT_DETECTION_SOURCE_IP)
NOTIFY(NAT_DETECTION_DESTINATION_IP)


IKEv2:(SESSION ID = 1,SA ID = 1):Insert SA

FlexClient#

IKEv2:(SESSION ID = 1,SA ID = 1):Retransmitting packet


IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 10.10.10.1:500/From
10.10.20.3:500/VRF i0:f0]

Initiator SPI : 502A6C10FFD6F8BC - Responder SPI : 0000000000000000 Message id: 0

IKEv2 IKE_SA_INIT Exchange REQUEST

Payload contents:

 SA KE N VID VID NOTIFY(NAT_DETECTION_SOURCE_IP)
NOTIFY(NAT_DETECTION_DESTINATION_IP)


IKEv2:(SESSION ID = 1,SA ID = 1):Retransmitting packet
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):Building packet for encryption.
Payload contents:
 VID IDi CERT CERTREQ NOTIFY(HTTP_CERT_LOOKUP_SUPPORTED) AUTH CFG SA TSi TSr
NOTIFY(INITIAL_CONTACT) NOTIFY(USE_TRANSPORT_MODE) NOTIFY(SET_WINDOW_SIZE)
NOTIFY(ESP_TFC_NO_SUPPORT) NOTIFY(NON_FIRST_FRAGS)

IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 10.10.10.1:500/From
10.10.50.5:500/VRF i0:f0]
Initiator SPI : DB0340BE40E55EE9 - Responder SPI : 756425ADB92B135A Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST
Payload contents:
 ENCR
IKEv2:(SESSION ID = 1,SA ID = 1):Retransmitting packet

IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 10.10.10.1:500/From
10.10.50.5:500/VRF i0:f0]
Initiator SPI : DB0340BE40E55EE9 - Responder SPI : 756425ADB92B135A Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST
Payload contents:
 ENCR

%LINK-3-UPDOWN: Interface Tunnel1, changed state to up
IKEv2:(SESSION ID = 1,SA ID = 1):Retransmitting packet

IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 10.10.10.1:500/From
10.10.50.5:500/VRF i0:f0]
Initiator SPI : DB0340BE40E55EE9 - Responder SPI : 756425ADB92B135A Message id: 1
IKEv2 IKE_AUTH Exchange REQUEST
Payload contents:
 ENCR
```

```
Router#show crypto ikev2 diagnose error
Exit Path Table - status: enable, current entry 4, deleted 0, max allow 50

Error(1): Failed to locate an item in the database

-Traceback= 9D64A28z 8CBE65Ez 8C66C40z 8CA6FF5z 8C746E0z 8C5359Az 8C5A25Ez
8C59F5Fz 8C97FE2z 8C979DBz 8C6A95Dz 8CC2BC3z

Error(1): A supplied parameter is incorrect

-Traceback= 9D64A28z 8CBE65Ez 8C66C40z 8CC273Ez 8CFA2F1z 8CF4038z 8C8B7F2z
8C7B455z 8C7AFE0z 8C5549Dz 8C5A25Ez 8C59F5Fz 8C6FDCAz 8C6EC98z 8C6A95Dz 8CC2BC3z

Error(1): AAA author not configured in IKEv2 profile

-Traceback= 9D64A28z 8CBE65Ez 8C66C40z 8CC6D2Dz 8CC6AC2z 8CC9661z 8CC9537z
8CC2B8Fz

Error(1): We requested a certificate, but the peer supplied none

-Traceback= 9D64A28z 8CBE65Ez 8C66C40z 8C7BABBz 8C7B6B8z 8C553F3z 8C5A25Ez
8C59F5Fz 8C97FE2z 8C979DBz 8C6A95Dz 8CC2BC3z
```

```
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Start PKI Session
IKEv2:(SA ID = 1):[PKI -> IKEv2] Starting of PKI Session PASSED
IKEv2:(SESSION ID = 1,SA ID = 1):[IKEv2 -> Crypto Engine] Computing DH public key,
DH Group 1
IKEv2:(SA ID = 1):[Crypto Engine -> IKEv2] DH key Computation PASSED
IKEv2:(SESSION ID = 1,SA ID = 1):Request queued for computation of DH key
IKEv2:IKEv2 initiator - no config data to send in IKE_SA_INIT exch
IKEv2:(SESSION ID = 1,SA ID = 1):Generating IKE_SA_INIT message
IKEv2:(SESSION ID = 1,SA ID = 1):IKE Proposal: 1, SPI size: 0
(initial negotiation),
Num. transforms: 4
   AES-CBC   SHA512   SHA512   DH_GROUP_768_MODP/Group 1

IKEv2:(SESSION ID = 1,SA ID = 1):Sending Packet [To 10.10.10.1:500/From
10.10.50.5:500/VRF i0:f0]
Initiator SPI : D007314789FFB97A - Responder SPI : 0000000000000000 Message id: 0
IKEv2 IKE_SA_INIT Exchange REQUEST
Payload contents:
 SA KE N VID VID NOTIFY(NAT_DETECTION_SOURCE_IP)
NOTIFY(NAT_DETECTION_DESTINATION_IP)

IKEv2:(SESSION ID = 1,SA ID = 1):Insert SA
```

```
IKEv2:(SESSION ID = 9,SA ID = 1):Processing IKE_SA_INIT message
IKEv2:(SESSION ID = 9,SA ID = 1):Received Policies: : Failed to find a matching
policyProposal 1:  AES-CBC-128 SHA512 SHA512 DH_GROUP_768_MODP/Group 1

IKEv2:(SESSION ID = 9,SA ID = 1):Expected Policies: : Failed to find a matching
policyProposal 1:  AES-CBC-128 SHA256 SHA256 DH_GROUP_2048_MODP/Group 14

Proposal 2:  AES-CBC-128 SHA256 SHA256 DH_GROUP_2048_MODP/Group 14

IKEv2:(SESSION ID = 9,SA ID = 1):: Failed to find a matching policy
IKEv2:(SESSION ID = 9,SA ID = 1):Sending no proposal chosen notify
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):Received Packet [From 10.10.10.1:500/To
10.10.50.5:500/VRF i0:f0]
Initiator SPI : D007314789FFB97A - Responder SPI : 63922088658CEA7B Message id: 0

IKEv2 IKE_SA_INIT Exchange RESPONSE
Payload contents:
 NOTIFY(NO_PROPOSAL_CHOSEN)
```

```
Router#show crypto ikev2 proposal

 IKEv2 proposal: default

      Encryption : AES-CBC-128

      Integrity  : SHA512

      PRF        : SHA512

      DH Group   : DH_GROUP_768_MODP/Group 1
```

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router (config)#crypto ikev2 proposal default
Router (config-ikev2-proposal)#integrity sha256
Router (config-ikev2-proposal)#group 14
Router #show crypto ikev2 proposal
 IKEv2 proposal: default
     Encryption : AES-CBC-128
     Integrity  : SHA256
     PRF        : SHA256
     DH Group   : DH_GROUP_2048_MODP/Group 14
```

```
IKEv2:(SESSION ID = 32,SA ID = 2):Get peer's authentication method
IKEv2:(SESSION ID = 32,SA ID = 2):: Peer's configured auth method mis-matches with
proposed auth method
IKEv2:(SESSION ID = 32,SA ID = 2):Verification of peer's authentication data
FAILED
IKEv2:(SESSION ID = 32,SA ID = 2):Sending authentication failure notify
IKEv2:(SESSION ID = 32,SA ID = 2):Building packet for encryption.
Payload contents:
 NOTIFY(AUTHENTICATION_FAILED)
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):Processing IKE_SA_INIT message

IKEv2:(SA ID = 1):[IKEv2 -> PKI] Retrieving trustpoint(s) from received
certificate hash(es)

IKEv2:(SA ID = 1):[PKI -> IKEv2] Retrieved trustpoint(s): NONE
IKEv2:% Received cert hash is invalid, using configured trustpoints from profile
for signing
```

```
FlexClient#show crypto pki trustpoints PKI status
Trustpoint PKI:
  Issuing CA certificate configured:
    Subject Name:
     cn=ca.cisco.com
    Fingerprint MD5: 8E628B19 7251C76A 543A76B0 8961EAEB
    Fingerprint SHA1: 46D32784 A5F7CF3F 2CC86337 6B0221E5 A5C5B4CD
  Router General Purpose certificate configured:
    Subject Name:
     hostname=flexclient.cisco.com
    Fingerprint MD5: DA640991 D2BA5328 F05594AE 09E2E235
    Fingerprint SHA1: 67C92EAB 24C7CDA1 B28E2696 1C5593D2 A23F0AB2
  State:
    Keys generated ..... Yes (General Purpose, non-exportable)
    Issuing CA authenticated ..... Yes
    Certificate request(s) ..... Yes
```

```
FlexClient#show crypto pki certificates PKI
Certificate
  Status: Available
  Certificate Serial Number (hex): 02
  Certificate Usage: General Purpose
  Issuer:
    cn=ca.cisco.com
  Subject:
    Name: flexclient.cisco.com
    hostname=flexclient.cisco.com
  Validity Date:
    start date: 19:03:45 JST Dec 21 2015
    end   date: 19:03:45 JST Dec 21 2016
```

```
  Associated Trustpoints: PKI
  Storage: nvram:s1#PKI.cer


CA Certificate
  Status: Available
  Certificate Serial Number (hex): 01
  Certificate Usage: Signature
  Issuer:
    cn=ca.cisco.com
  Subject:
    cn=ca.cisco.com
  Validity Date:
    start date: 19:02:16 JST Dec 21 2015
    end   date: 19:02:16 JST Dec 21 2018
  Associated Trustpoints: 20 PKI
  Storage: nvram:s1#1CA.cer
```

```
FlexClient#show crypto pki certificates verbose
Certificate
  Status: Available
  Version: 3
  Certificate Serial Number (hex): 02
  …
  X509v3 extensions:
    X509v3 Key Usage: A0000000
      Digital Signature
      Key Encipherment
    X509v3 Subject Key ID: 193A5E31 B1B5DECB C1172311 A61FEF7E E049CED0
```

Since we are talking about using the public key to validate a signature, if the KeyUsage extension is present, then at least one of the digitalSignature or the nonRepudiation bits in the KeyUsage extension MUST be set (both can be set as well). It is also fine if other KeyUsage bits are set.

```
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Retrieving trustpoint(s) from received
certificate hash(es)
IKEv2:(SA ID = 1):[PKI -> IKEv2] Retrieved trustpoint(s): NONE
IKEv2:% Received cert hash is invalid, using configured trustpoints from profile
for signing

IKEv2:(SA ID = 1):[IKEv2 -> PKI] Getting cert chain for the trustpoint 1
IKEv2:(SA ID = 1):[PKI -> IKEv2] Getting of cert chain for the trustpoint PASSED
IKEv2:(SESSION ID = 1,SA ID = 1):Get peer's authentication method
IKEv2:(SESSION ID = 1,SA ID = 1):Peer's authentication method is 'RSA'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Validating certificate chain
IKEv2:(SA ID = 1):[PKI -> IKEv2] Validation of certificate chain FAILED
IKEv2:(SESSION ID = 1,SA ID = 1):: Platform errors
IKEv2:(SESSION ID = 1,SA ID = 1):Verify cert failed
IKEv2:(SESSION ID = 1,SA ID = 1):Verification of peer's authentication data FAILED
IKEv2:(SESSION ID = 1,SA ID = 1):Sending authentication failure notify
IKEv2:(SESSION ID = 1,SA ID = 1):Building packet for encryption.
Payload contents:
 NOTIFY(AUTHENTICATION_FAILED)
```

IKEv2:Current time is more than cert validity time

IKEv2:(SESSION ID = 32,SA ID = 1):Failed to set IKE policy: Failed to build certificate payload

IKEv2:(SESSION ID = 32,SA ID = 1):Verification of peer's authentication data FAILED

IKEv2:(SESSION ID = 32,SA ID = 1):Sending authentication failure notify

IKEv2:(SESSION ID = 32,SA ID = 1):Building packet for encryption.

Payload contents:

NOTIFY(AUTHENTICATION_FAILED)

```
FlexClient#show ntp associations

  address         ref clock       st   when   poll reach  delay  offset   disp
*~10.10.10.1      127.127.1.1      8     17     64     1  0.000   0.000 189.44
 * sys.peer, # selected, + candidate, - outlyer, x falseticker, ~ configured
```

```
FlexClient#debug ntp packet
NTP packets debugging is on

NTP message sent to 10.10.10.1, from interface 'Ethernet0/0' (10.10.20.3).
NTP message received from 10.10.10.1 on interface 'Ethernet0/0' (10.10.20.3).
```

```
FlexClient#show ntp associations

  address         ref clock       st   when   poll reach  delay  offset   disp
 ~10.10.10.1      .INIT.          16      -     64     0  0.000   0.000 15937.
 * sys.peer, # selected, + candidate, - outlyer, x falseticker, ~ configured

FlexClient#debug ntp packet
NTP packets debugging is on

NTP message sent to 10.10.10.1, from interface 'NULL' (0.0.0.0).
```

```
crypto pki trustpoint pki
match certificate cert_map allow expired-certificate
```

```
IKEv2:(SESSION ID = 39,SA ID = 1):Searching policy based on peer's identity
'hostname=flexclient.cisco.com' of type 'DER ASN1 DN'

IKEv2:Optional profile description not updated in PSH

IKEv2:Searching Policy with fvrf 0, local address 10.10.10.1

IKEv2:Using the Default Policy for Proposal

IKEv2:Found Policy 'default'

IKEv2:Found matching IKEv2 profile 'default'
```

```
IKEv2:(SESSION ID = 40,SA ID = 1):Searching policy based on peer's identity
'hostname=flexclient.cisco.com' of type 'DER ASN1 DN'

IKEv2:% IKEv2 profile not found

IKEv2:(SESSION ID = 40,SA ID = 1):: Failed to locate an item in the database

IKEv2:(SESSION ID = 40,SA ID = 1):Verification of peer's authentication data
FAILED

IKEv2:(SESSION ID = 40,SA ID = 1):Sending authentication failure notify

IKEv2:(SESSION ID = 40,SA ID = 1):Building packet for encryption.
```

```
Router#show running-config | section certificate map
crypto pki certificate map certmap 10
 subject-name eq hostname=flexclient.cisco.com
```

```
Router(config)#crypto ikev2 profile test
```

IKEv2 profile MUST have:

1. A local and a remote authentication method.
2. A match identity or a match certificate or match any statement.

```
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Retrieving trustpoint(s) from received
certificate hash(es)
IKEv2:(SA ID = 1):[PKI -> IKEv2] Retrieved trustpoint(s): 'pki'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Getting cert chain for the trustpoint pki
IKEv2:(SA ID = 1):[PKI -> IKEv2] Getting of cert chain for the trustpoint PASSED
IKEv2:(SESSION ID = 5,SA ID = 1):Get peer's authentication method
IKEv2:(SESSION ID = 5,SA ID = 1):Peer's authentication method is 'RSA'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Validating certificate chain
IKEv2:(SA ID = 1):[PKI -> IKEv2] Validation of certificate chain FAILED
IKEv2:(SESSION ID = 5,SA ID = 1):
%PKI-6-PKI_CRL_DOWNLOADED: CRL download notification sent for Issuer = cn=sub.
IKEv2:(SESSION ID = 5,SA ID = 1):Save pubkey
IKEv2:(SESSION ID = 5,SA ID = 1):Verify peer's authentication data
IKEv2:[IKEv2 -> Crypto Engine] Generate IKEv2 authentication data
IKEv2:[Crypto Engine -> IKEv2] IKEv2 authentication data generation PASSED
IKEv2:(SA ID = 1):[IKEv2 -> Crypto Engine] Verify signed authentication data
IKEv2:(SA ID = 1):[Crypto Engine -> IKEv2] Verification of signed authentication
data PASSED
```

```
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Getting cert chain for the trustpoint pki
IKEv2:(SA ID = 1):[PKI -> IKEv2] Getting of cert chain for the trustpoint PASSED
IKEv2:(SESSION ID = 8,SA ID = 1):Get peer's authentication method
IKEv2:(SESSION ID = 8,SA ID = 1):Peer's authentication method is 'RSA'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Validating certificate chain
IKEv2:(SA ID = 1):[PKI -> IKEv2] Validation of certificate chain FAILED
IKEv2:(SESSION ID = 8,SA ID = 1):
Router#
IKEv2:: Negotiation context locked currently in use
Router#
%PKI-3-SOCKETSELECT: Failed to select the socket.
IKEv2:(SESSION ID = 8,SA ID = 1):Verify cert failed
IKEv2:(SESSION ID = 8,SA ID = 1):Verification of peer's authentication data FAILED
IKEv2:(SESSION ID = 8,SA ID = 1):Sending authentication failure notify
IKEv2:(SESSION ID = 8,SA ID = 1):Building packet for encryption.
Payload contents:
 NOTIFY(AUTHENTICATION_FAILED)
```

```
CRYPTO_PKI: (A0011) Requesting CRL from SCEP
CRYPTO_PKI: locked trustpoint pki, refcount is 2
CRYPTO_PKI: http connection opened via VRF OUTSIDE
CRYPTO_PKI: Sending HTTP message

CRYPTO_PKI: Reply HTTP header:
HTTP/1.0
Host: 10.10.30.3

Router#
IKEv2:: Negotiation context locked currently in use
Router#
%PKI-3-SOCKETSELECT: Failed to select the socket.
```

```
tcp0: O CLOSED 10.10.30.3:80 10.10.10.1:48085 seq 3276603876
        OPTS 4 SYN  WIN 4128
tcp0: R SYNSENT 10.10.30.3:80 10.10.10.1:48085 seq 3276603876
        OPTS 4 SYN  WIN 4128
TCP0: timeout #1 - timeout is 4000 ms, seq 3276603876
TCP: (48085) -> 10.10.30.3(80)
```

```
R1#show crypto pki counters
PKI Sessions Started: 50
PKI Sessions Ended: 50
PKI Sessions Active: 0
Successful Validations: 4
Failed Validations: 46
Bypassed Validations: 0
Pending Validations: 0
CRLs checked: 56
CRL - fetch attempts: 49
CRL - failed attempts: 45
CRL - rejected busy fetching: 1
AAA authorizations: 0
```

```
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Retrieving trustpoint(s) from received
certificate hash(es)
IKEv2:(SA ID = 1):[PKI -> IKEv2] Retrieved trustpoint(s): 'pki'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Getting cert chain for the trustpoint pki
IKEv2:(SA ID = 1):[PKI -> IKEv2] Getting of cert chain for the trustpoint PASSED
IKEv2:(SESSION ID = 200,SA ID = 1):Get peer's authentication method
IKEv2:(SESSION ID = 200,SA ID = 1):Peer's authentication method is 'RSA'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Validating certificate chain
IKEv2:(SA ID = 1):[PKI -> IKEv2] Validation of certificate chain PASSED
IKEv2:(SESSION ID = 200,SA ID = 1):Save pubkey
IKEv2:(SESSION ID = 200,SA ID = 1):Verify peer's authentication data
IKEv2:[IKEv2 -> Crypto Engine] Generate IKEv2 authentication data
IKEv2:[Crypto Engine -> IKEv2] IKEv2 authentication data generation PASSED
IKEv2:(SA ID = 1):[IKEv2 -> Crypto Engine] Verify signed authentication data
IKEv2:(SA ID = 1):[Crypto Engine -> IKEv2] Verification of signed authentication
data PASSED
```

```
Router(config-ikev2-profile)#pki trustpoint 1 sign
```

```
Router(config-ikev2-profile)#pki trustpoint 1 verify
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):Processing IKE_SA_INIT message

IKEv2:(SA ID = 1):[IKEv2 -> PKI] Retrieving trustpoint(s) from received
certificate hash(es)

IKEv2:(SA ID = 1):[PKI -> IKEv2] Retrieved trustpoint(s): NONE

IKEv2:% Received cert hash is invalid, using configured trustpoints from profile
for signing
```

```
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Retrieving trustpoint(s) from received
certificate hash(es)
IKEv2:(SA ID = 1):[PKI -> IKEv2] Retrieved trustpoint(s): NONE
IKEv2:% Received cert hash is invalid, using configured trustpoints from profile
for signing

IKEv2:(SA ID = 1):[IKEv2 -> PKI] Getting cert chain for the trustpoint 1
IKEv2:(SA ID = 1):[PKI -> IKEv2] Getting of cert chain for the trustpoint PASSED
IKEv2:(SESSION ID = 1,SA ID = 1):Get peer's authentication method
IKEv2:(SESSION ID = 1,SA ID = 1):Peer's authentication method is 'RSA'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Validating certificate chain
IKEv2:(SA ID = 1):[PKI -> IKEv2] Validation of certificate chain FAILED
IKEv2:(SESSION ID = 1,SA ID = 1):: Platform errors
IKEv2:(SESSION ID = 1,SA ID = 1):Verify cert failed
IKEv2:(SESSION ID = 1,SA ID = 1):Verification of peer's authentication data FAILED
IKEv2:(SESSION ID = 1,SA ID = 1):Sending authentication failure notify
IKEv2:(SESSION ID = 1,SA ID = 1):Building packet for encryption.
Payload contents:
 NOTIFY(AUTHENTICATION_FAILED)
```

```
map_db_check_isakmp_profile profile did not match
map_db_find_best did not find matching map
IPSEC(ipsec_process_proposal): proxy identities not supported
IKEv2:(SA ID = 1):There was no IPSEC policy found for received TS
IKEv2:(SA ID = 1):Sending TS unacceptable notify
```

```
Routert#show running-config | section ikev2 keyring
crypto ikev2 keyring key1
 peer 10.10.10.1
  address 10.10.10.1
  pre-shared-key cisco
```

```
IKEv2:% Getting preshared key from profile keyring key1
IKEv2:% key not found.
IKEv2:Failed to initiate sa
```

```
IKEv2:(SESSION ID = 12,SA ID = 1):Peer's authentication method is 'PSK'
IKEv2:(SESSION ID = 12,SA ID = 1):Get peer's preshared key for flexclient.cisco.
com
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Password request sent
IKEv2:IKEv2 local AAA - user author request for 'flexclient'
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received password response
```

```
Router1#show run | section ikev2 keyring
crypto ikev2 keyring key1
 peer 10.10.10.1
  address 10.10.10.1
  pre-shared-key cisco

Router2#show run | section ikev2 keyring
crypto ikev2 keyring key1
 peer 10.10.10.100
  address 10.10.10.100
  pre-shared-key hex 636973636f
```

```
Router#show crypto pki certificates verbose

…

X509v3 extensions:
     X509v3 Key Usage: 86000000
       Digital Signature
       Key Cert Sign
       CRL Signature
    X509v3 extensions:
   X509v3 Extended Key Usage:
     TLS Web Server Authentication
     TLS Web Client Authentication
```

```
Router#show run | section radius
aaa group server radius radius_group
 server-private 10.10.20.1 auth-port 1812 acct-port 1813 key cisco123
 aaa authentication login rad1 group radius_group
```

```
router#test aaa group radius_group test lab new-code

RADIUS/ENCODE(00000000):Orig. component type = INVALID RADIUS/ENCODE(00000000):
dropping service type, "radius-server attribute 6 on-for-login-auth" is off
RADIUS(00000000): Config NAS IP: 10.10.10.1
RADIUS(00000000): sending
RADIUS/ENCODE: Best Local IP-Address 10.10.10.1 for Radius-Server 10.10.20.1
RADIUS(00000000): Send Access-Request to 10.10.20.1:1812 id 1812/1, len 50
RADIUS:  authenticator CA DB F4 9B 7B 66 C8 A9 - D1 99 4E 8E A4 46 99 B4
RADIUS:  User-Password       [2]   18  *
RADIUS:  User-Name           [1]   6   "test"
RADIUS:  NAS-IP-Address      [4]   6   10.10.10.1
RADIUS: Received from id 1812/1 10.10.20.1:1812, Access-Accept, len 44
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):Building packet for encryption.
Payload contents:
 VID IDi CERTREQ NOTIFY(HTTP_CERT_LOOKUP_SUPPORTED) CFG SA TSi TSr
NOTIFY(INITIAL_CONTACT) NOTIFY(USE_TRANSPORT_MODE) NOTIFY(SET_WINDOW_SIZE)
NOTIFY(ESP_TFC_NO_SUPPORT) NOTIFY(NON_FIRST_FRAGS)
```

```
IKEv2:Cannot use IP address as EAP identity
IKEv2:Received response from authenticator
IKEv2:(1): Authenticator sent NULL EAP message
IKEv2:(1): Extensible Authentication Protocol failed
IKEv2:(1): Auth exchange failed
IKEv2:(1): Abort exchange
```

```
IKEv2:Exchange type: IKE_AUTH, flags: INITIATOR
IKEv2:Message id: 0x2, length: 68
REAL Decrypted packet:
    00 00 00 0E 02 3B 00 0A 01 63 69 73 63 6F
EAP  Next payload: NONE, reserved: 0x0, length: 14
Code: response: id: 59, length: 10
Type: identity
```

```
IKEv2:(1): Processing EAP response

RADIUS/ENCODE(00000013):Orig. component type = VPN IPSEC

RADIUS:  AAA Unsupported Attr: service-type      [345] 4   1 RADIUS/
ENCODE(00000013): dropping service type, "radius-server attribute 6
on-for-login-auth" is off

RADIUS(00000013): Config NAS IP: 0.0.0.0

RADIUS(00000013): Config NAS IPv6: ::

RADIUS/ENCODE(00000013): acct_session_id: 9

RADIUS(00000013): sending

RADIUS/ENCODE: Best Local IP-Address 10.10.10.10.1 for Radius-Server 10.10.10.100

RADIUS(00000013): Send Access-Request to 10.10.10.10.100:1812 id 1645/10, len 82

RADIUS:  authenticator 10 16 B6 71 69 BE 3C 8F - D4 9B 77 8B AF 5A 9E ED

RADIUS:  Service-Type        [6]   6   Login                      [1] RADIUS:
Calling-Station-Id  [31]  13  "10.10.20.100"

RADIUS:  User-Name           [1]   7   "cisco"

RADIUS:  EAP-Message         [79]  12

RADIUS:   02 3B 00 0A 01 63 69 73 63 6F            [ ;cisco]

RADIUS:  Message-Authenticato[80]  18   RADIUS:   4F DE 44 97 7B 04 0E A2 C9 FC 55
FA 72 2F 53 34          [ OD{Ur/S4]

RADIUS:  NAS-IP-Address      [4]   6   10.10.10.1
RADIUS(00000013): Sending a IPv4 Radius Packet
```

```
RADIUS: Received from id 1812/30 10.10.10.100:1812, Access-Challenge, len 98
RADIUS:  authenticator 02 E3 BC 5C 7B EF DC 38 - 0C 14 12 4A 4F A0 6C 08
RADIUS:  Framed-IP-Address    [8]   6   172.16.1.1
RADIUS:  Service-Type         [6]   6   Framed                    [2]
RADIUS:  Framed-Protocol      [7]   6   PPP                       [1]
RADIUS:  EAP-Message          [79]  24
RADIUS:   01 3C 00 16 04 10 67 30 CF 35 9B 2E A0 56 85 F4 51 7B 0F 01 7F 6B
[ <g05.VQ{k]
RADIUS:  Message-Authenticato[80]  18
RADIUS:   E1 11 71 E1 DC 3A DB 30 89 3D 62 A8 F0 FC 63 FB            [ q:0=bc]
RADIUS:  State                [24]  18
RADIUS:   16 47 B3 FF 16 7B B7 6A 0B 77 42 E6 F7 FC 0A 7B            [ G{jwB{]
RADIUS(00000013): Received from id 1811/30
RADIUS/DECODE: EAP-Message fragments, 22, total 22 bytes
RADIUS: Constructed " ppp negotiate"
IKEv2:(SA ID = 1):[AAA -> IKEv2] Successful response received
IKEv2:Received response from authenticator
IKEv2:(SESSION ID = 5,SA ID = 1):Generating EAP request
IKEv2:(SESSION ID = 5,SA ID = 1):Building packet for encryption.
Payload contents:
 EAP


IKEv2:(SESSION ID = 5,SA ID = 1):Sending Packet [To 10.10.10.10:500/From
10.10.10.1:500/VRF i0:f0]
Initiator SPI : CEC6599A1DDEDAD4 - Responder SPI : 7044B33369C2FC4A Message id: 2
IKEv2 IKE_AUTH Exchange RESPONSE
Payload contents:
 ENCR


…
IKEv2:(SA ID = 1):[AAA -> IKEv2] Successful response received
IKEv2:Received response from authenticator
IKEv2:(SESSION ID = 5,SA ID = 1):Generating EAP request
IKEv2:(SESSION ID = 5,SA ID = 1):Building packet for encryption.
Payload contents:
 EAP
```

authentication remote eap timeout <seconds>

```
IKEv2:(SA ID = 1):[IKEv2 -> Crypto Engine] Verify signed authentication data

IKEv2:(SA ID = 1):[Crypto Engine -> IKEv2] Verification of signed authentication
data PASSED

IKEv2:(SESSION ID = 17,SA ID = 1):Processing INITIAL_CONTACT

IKEv2:Using mlist default and username service1 for group author request

AAA/BIND(0000002B): Bind i/f

AAA/AUTHOR (0x2B): Pick method list 'default'

IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorisation request sent

IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response

IKEv2:(SESSION ID = 17,SA ID = 1):Received valid config mode data
```

```
aaa authorization group cert list default default local
```

```
IKEv2:Error constructing config reply
IKEv2:(SESSION ID = 31,SA ID = 1):Error in retrieving config mode data to send
IKEv2:(SESSION ID = 31,SA ID = 1):Fail to alloc IP addr
```

```
aaa attribute list service1
 attribute type interface-config "ip ccess-group service1 in" protocol ip
```

```
IKEv2:AAA clone request failed, error 7
IKEv2:(SESSION ID = 0,SA ID = 1):Error while applying per user attributes
```

```
IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorisation request sent
AAA/LOCAL/AUTHEN: starting
AAA/LOCAL/AUTHEN(17): authorizing crypto service1
IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response
IKEv2:(SESSION ID = 7,SA ID = 1):Received valid config mode data
IKEv2:Config data received:
```

```
crypto ikev2 profile default
 aaa authorization user {psk|eap} cached
```

```
VT[Vi1]:Applying config commands on process "VTEMPLATE Background Mgr" (373)
 VT[Vi1]:default ip mtu 1420
 VT[Vi1]:default ip tcp adjust-mss 1400
 VT[Vi1]:default ip access-group subscriber1 in
 VT[Vi1]:default ip verify unicast source reachable-via rx
 VT[Vi1]:default ip unreachables
 VT[Vi1]:default ip unnumbered loopback1
 VT[Vi1]:default ip access-group service1 in
 VT[Vi1]:end
 VT[Vi1]:Remove cloneblk AAA from vaccess with vtemplate/AAA
 VT[Vi1]:Unclone vaccess, 3 command(s) to be removed
 VT[Vi1]:default tunnel protection ipsec profile default
 VT[Vi1]:default tunnel vrf OUTSIDE
 VT[Vi1]:default tunnel source Ethernet0/0
 VT[Vi1]:end
```

```
Router#show crypto ipsec transform-set
Transform set default: { esp-aes esp-sha-hmac  }
   will negotiate = { Transport,  },

Transform set aes-sha256: { esp-aes esp-sha256-hmac  }
   will negotiate = { Transport,  },
```

```
Router#show crypto ipsec profile
IPSEC profile default
     IKEv2 Profile: default
     Security association lifetime: 4608000 kilobytes/3600 seconds
     Responder-Only (Y/N): N
     PFS (Y/N): Y
     DH group:  group14
     Mixed-mode : Disabled
     Transform sets={
          aes-sha256:  { esp-aes esp-sha256-hmac  } ,
```

```
IKEv2:(SESSION ID = 473,SA ID = 1):Set received config mode data

IKEv2:(SESSION ID = 473,SA ID = 1):Processing IKE_AUTH message

IKEv2:% DVTI create request sent for profile default with PSH index 1.

IKEv2:(SESSION ID = 473,SA ID = 1):

%LINEPROTO-5-UPDOWN: Line protocol on Interface Virtual-Access1, changed state
to down

IKEv2:% DVTI Vi1 created for profile default with PSH index 1.

IKEv2:IPSec policy validate request sent for profile default with psh index 1.

IKEv2:(SA ID = 1):[IPsec -> IKEv2] Callback received for the validate proposal -
FAILED.

IKEv2:(SESSION ID = 473,SA ID = 1):Received Policies: : Failed to find a matching
policyESP: Proposal 1:  3DES SHA256 Don't use ESN

IKEv2:(SESSION ID = 473,SA ID = 1):Expected Policies: : Failed to find a matching
policy

IKEv2:(SESSION ID = 473,SA ID = 1):: Failed to find a matching policy

IKEv2:(SESSION ID = 473,SA ID = 1):Sending no proposal chosen notify

***

IKEv2:(SESSION ID = 473,SA ID = 1):Building packet for encryption.

Payload contents:

 VID IDr CERT AUTH NOTIFY(NO_PROPOSAL_CHOSEN)
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):Received Packet [From 10.10.10.1:500/To
10.10.50.5:500/VRF i0:f0]
Initiator SPI : 58C298669EA6A95F - Responder SPI : 38F270737BD3EE39 Message id: 1
IKEv2 IKE_AUTH Exchange RESPONSE
Payload contents:
 VID IDr CERT AUTH NOTIFY(NO_PROPOSAL_CHOSEN)

IKEv2:(SESSION ID = 1,SA ID = 1):Process auth response notify
…
IKEv2:(SESSION ID = 1,SA ID = 1):Verify peer's policy
IKEv2:(SESSION ID = 1,SA ID = 1):Peer's policy verified
IKEv2:(SESSION ID = 1,SA ID = 1):Get peer's authentication method
IKEv2:(SESSION ID = 1,SA ID = 1):Peer's authentication method is 'RSA'
IKEv2:(SA ID = 1):[IKEv2 -> PKI] Validating certificate chain
IKEv2:(SA ID = 1):[PKI -> IKEv2] Validation of certificate chain PASSED
IKEv2:(SESSION ID = 1,SA ID = 1):Save pubkey
IKEv2:(SESSION ID = 1,SA ID = 1):Verify peer's authentication data
IKEv2:[IKEv2 -> Crypto Engine] Generate IKEv2 authentication data
```

```
IKEv2:[Crypto Engine -> IKEv2] IKEv2 authentication data generation PASSED

IKEv2:(SA ID = 1):[IKEv2 -> Crypto Engine] Verify signed authentication data

IKEv2:(SA ID = 1):[Crypto Engine -> IKEv2] Verification of signed authentication
data PASSED

IKEv2:(SESSION ID = 1,SA ID = 1):Check for EAP exchange

IKEv2:Using mlist default and username default for group author request

IKEv2:(SA ID = 1):[IKEv2 -> AAA] Authorisation request sent

IKEv2:(SA ID = 1):[AAA -> IKEv2] Received AAA authorisation response

IKEv2:(SA ID = 1):[IKEv2 -> PKI] Close PKI Session

IKEv2:(SA ID = 1):[PKI -> IKEv2] Closing of PKI Session PASSED

IKEv2:(SESSION ID = 1,SA ID = 1):IKEV2 SA created; inserting SA into database.
SA lifetime timer (86400 sec) started

IKEv2:(SESSION ID = 1,SA ID = 1):Session with IKE ID PAIR (hostname=R1.cisco.com,
hostname=R5.cisco.com,cn=r5,o=cisco,ou=service1) is UP

IKEv2:IKEv2 MIB tunnel started, tunnel index 1

IKEv2:(SESSION ID = 1,SA ID = 1):Checking for duplicate IKEv2 SA

IKEv2:(SESSION ID = 1,SA ID = 1):No duplicate IKEv2 SA found

IKEv2:(SESSION ID = 1,SA ID = 1):Queuing IKE SA delete request reason: unknown

IKEv2:(SESSION ID = 1,SA ID = 1):Sending DELETE INFO message for IPsec SA [SPI:
0xAEF5E4BE]

IKEv2:(SESSION ID = 1,SA ID = 1):Building packet for encryption.

Payload contents:

 DELETE
```

```
IKEv2:(SESSION ID = 1067,SA ID = 1):Processing IKE_AUTH message
IKEv2:% DVTI create request sent for profile default with PSH index 1.
IKEv2:(SESSION ID = 1067,SA ID = 1):
%LINEPROTO-5-UPDOWN: Line protocol on Interface Virtual-Access1, changed state
to down
IKEv2:% DVTI Vi1 created for profile default with PSH index 1.
IKEv2:IPSec policy validate request sent for profile default with psh index 1.
IKEv2:(SA ID = 1):[IPsec -> IKEv2] Callback received for the validate proposal -
FAILED.
IKEv2:(SESSION ID = 1067,SA ID = 1):: There was no IPSEC policy found for
received TS
IKEv2:(SESSION ID = 1067,SA ID = 1):Sending TS unacceptable notify
```

```
SA  Next payload: TSi, reserved: 0x0, length: 44
  last proposal: 0x0, reserved: 0x0, length: 40
  Proposal: 1, Protocol id: ESP, SPI size: 4, #trans: 3    last transform: 0x3,
reserved: 0x0: length: 12
    type: 1, reserved: 0x0, id: AES-CBC
    last transform: 0x3, reserved: 0x0: length: 8
    type: 3, reserved: 0x0, id: SHA256
    last transform: 0x0, reserved: 0x0: length: 8
    type: 5, reserved: 0x0, id: Don't use ESN
 TSi  Next payload: TSr, reserved: 0x0, length: 24
    Num of TSs: 1, reserved 0x0, reserved 0x0
    TS type: TS_IPV4_ADDR_RANGE, proto id: 0, length: 16
    start port: 0, end port: 65535
    start addr: 0.0.0.0, end addr: 255.255.255.255
 TSr  Next payload: NOTIFY, reserved: 0x0, length: 24
    Num of TSs: 1, reserved 0x0, reserved 0x0
    TS type: TS_IPV4_ADDR_RANGE, proto id: 0, length: 16
    start port: 0, end port: 65535
    start addr: 0.0.0.0, end addr: 255.255.255.255
```

%CRYPTO-4-PKT_REPLAY_ERR: decrypt: replay check failed connection id=#, sequence number=#

```
%CRYPTO-4-PKT_REPLAY_ERR: decrypt: replay check failed
        connection id=771, sequence number=1193
```

```
Router#show crypto ipsec sa | in peer|conn id
   current_peer 10.1.1.1 port 500
        conn id: 771, flow_id: SW:771, sibling_flags 80000046, crypto-map:
Tunnel0-head-0
        conn id: 772, flow_id: SW:772, sibling_flags 80000046, crypto-map:
Tunnel0-head-0
```

```
%IPSEC-3-REPLAY_ERROR: IPSec SA receives anti-replay error, DP Handle 6,
          src_addr 10.10,10.1, dest_addr 10.10.10.2, SPI 0x1a2b3c4d
```

```
Router#debug platform condition ipv4 10.2.0.200/32 ingress
Router#debug platform condition start
```

```
Router#debug platform packet enable
Router#debug platform packet-trace packet 64
Router#debug platform packet-trace copy packet input l3 size 100
```

```
Router#show platform packet-trace summary
Pkt    InputOutput    State  Reason
0 Gi1/0/0   Tu1   CONS    Packet Consumed
1 Gi1/0/0   Tu1   CONS    Packet Consumed
2 Gi1/0/0   Tu1   DROP    053 (IpsecInput)
3 Gi1/0/0   Tu1   DROP    053 (IpsecInput)
4 Gi1/0/0   Tu1   CONS    Packet Consumed
```

```
Router#show platform packet-trace packet 2
Packet: 2    CBUG ID: 2
Summary
  Input : GigabitEthernet1/0/0
  Output: Tunnel1
  State : DROP 053 (IpsecInput)
  Timestamp : 3233497953773
Path Trace
  Feature: IPV4
Source   : 10.1.1.1
Destination : 10.1.1.2
Protocol: 50 (ESP)
  Feature: IPSec
Action: DECRYPT
SA Handle : 3
SPI    : 0x4c1d1e90
Peer Addr : 10.1.1.1
Local Addr: 10.1.1.2
  Feature: IPSec
Action: DROP
Sub-code  : 019 - CD_IN_ANTI_REPLAY_FAIL
Packet Copy In
  45000428 00110000 fc329575 0a0200c8 0a010064 4c1d1e90 000000df 790aa252
  e9951cd9 57024433 d97c7cb8 58e0c869 2101f1ef 148c2a12 f309171d 1b7a4771
  d8868af7 7bae9967 7d880197 46c6a079 d0143e43 c9024c61 0045280a d57b2f5e
  23f06bc3 ab6b6b81 c1b17936 98939509 7aec966e 4dd848d2 60517162 9308ba5d
```

```
Router#show interface tunnel 1
Tunnel1 is up, line protocol is down
  Hardware is Tunnel
  …
 interface <OK>
  Tunnel protocol/transport GRE/IP
    Key 0x2, sequencing disabled
    Checksumming of packets disabled
```

```
ip route 192.168.0.0 255.255.0.0 Tunnel1
```

```
Router#show crypto session brief
Status: A- Active, U - Up, D - Down, I - Idle, S - Standby, N - Negotiating
        K - No IKE
ivrf = (none)
            Peer     I/F        Username          Group/Phase1_id   Uptime Stat
     10.10.60.6     Vi2                  hostname=Router2.cisco.com,cn 00:01:01
UA
     10.10.50.5     Vi1                  hostname=Router1.cisco.com,cn 00:31:38
UA
```

```
Router#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
2         10.10.10.1/500        10.10.50.5/500        OUTSIDE/none         READY
      Encr: AES-CBC, keysize: 128, PRF: SHA256, Hash: SHA256, DH Grp:14, Auth
sign: RSA, Auth verify: RSA
      Life/Active Time: 86400/220 sec
      CE id: 1026, Session-id: 19
      Status Description: Negotiation done
      Local spi: B6D1DDD763BE26D7       Remote spi: 8639ADD29DBB80B2
      …
      Assigned host addr: 192.168.1.14
      Initiator of SA : No
```

```
Router#show ip route 192.168.1.14

% Subnet not in table
```

```
IKEv2:(SESSION ID = 1,SA ID = 1):Config-type: Config-set
IKEv2:(SESSION ID = 1,SA ID = 1):Attrib type: app-version, length: 256, data:
Cisco IOS Software,
..
```

```
Router1#show crypto ikev2 authorization policy
 IKEv2 Authorization Policy : default
  route accept any tag : 1 distance : 1
```

```
Router#ping 192.168.1.1 source tunnel 1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.1.19

Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/5 ms
```

```
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to up
%TUN-5-RECURDOWN: Tunnel0 temporarily disabled due to recursive routing
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel0, changed state to down
%LINEPROTO-5-UPDOWN:
```

```
Router(config)#crypto ipsec security-association dummy pps ?
  <0-25>  Rate of simulated traffic (in PPS)

Router(config)#crypto ipsec security-association dummy seconds ?
  <1-3600>  Rate of simulated traffic (seconds between packets)
```

```
Router#show interfaces FastEthernet4
FastEthernet4 is up, line protocol is up
  Hardware is PQUICC_FEC, address is 0021.5528.7f84 (bia 0021.5528.7f84)
  Internet address is 10.48.66.98/23
  MTU 1500 bytes, BW 100000 Kbit/sec, DLY 100 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation 802.1Q Virtual LAN, Vlan ID  1., loopback not set
  Keepalive set (10 sec)
```

```
Router#show running-config interface FastEthernet4
interface FastEthernet4
 ip address 10.0.0.1 255.255.255.0
 ip mtu 1400
 ipv6 enable
 ipv6 mtu 1300
end

Router#show ip interface FastEthernet4
FastEthernet4 is up, line protocol is up
  Internet address is 10.0.0.1/24
  Broadcast address is 255.255.255.255
  Address determined by setup command
  MTU is 1400 bytes
  ...

Router#show ipv6 interface FastEthernet4
FastEthernet4 is up, line protocol is up
  IPv6 is enabled, link-local address is FE80::221:55FF:FE28:7F84
  No Virtual link-local address(es):
  No global unicast address is configured
  Joined group address(es):
    FF02::1
    FF02::1:FF28:7F84
  MTU is 1300 bytes
  ...
```

```
Router#show ip traffic | include reassembled|fragmented
  Frags: 7 reassembled, 0 timeouts, 0 couldn't reassemble
         7 fragmented, 14 fragments, 0 couldn't fragment
```

```
Router#show ipv6 traffic | include reassembled|fragmented
          17 fragments, 8 total reassembled
          10 fragmented into 20 fragments, 0 failed
```

```
interface FastEthernet0/0
 ip address 209.165.201.1 255.255.255.252
 ip tcp adjust-mss 1416
 ipv6 address 2001:DB8:FFF1::10/127
 ipv6 enable
```

```
Router#show crypto ipsec sa | include mtu
      path mtu 1400, ip mtu 1500, ip mtu idb Ethernet0/0
```

```
Router#show crypto ipsec sa | include mtu
     plaintext mtu 1342, path mtu 1400, ip mtu 1500, ip mtu idb Ethernet0/0
```

```
Router#show crypto ipsec sa | include mtu
     path mtu 1400, ipv6 mtu 1500, ipv6 mtu idb Ethernet0/0

Router#show ipv6 mtu
 MTU      Since     Source Address      Destination Address
1400     00:00:11  2001:DB8:FFF1::10   2001:DB8:FFF1::2510
```

```
Router#show running-config interface Tunnel0
interface Tunnel0
 ip address 10.0.0.1 255.255.255.252
 tunnel source Loopback1
 tunnel mode ipsec ipv4
 tunnel destination 209.165.200.225
 tunnel protection ipsec profile default
end

Router#show crypto ipsec sa | include mtu
     path mtu 1500, ip mtu 1500, ip mtu idb Ethernet0/0

Router#show interface Tunnel0 | include MTU
  MTU 17878 bytes, BW 100 Kbit/sec, DLY 50000 usec,
  Tunnel transport MTU 1438 bytes

Router#show ip interface Tunnel0 | include MTU
  MTU is 1438 bytes

Router#show cef interface Tunnel0 | include output interface
  Real output interface is Ethernet0/0
```

```
Router#show running-config interface Tunnel0
interface Tunnel0
 ip address 10.0.0.1 255.255.255.252
 tunnel source Ethernet0/0
 tunnel mode gre ipv6
 tunnel destination 2001:DB8:FFF1::2510
 tunnel key 1
end

Router#show interface Tunnel0 | include MTU
   MTU 1452 bytes, BW 100 Kbit/sec, DLY 50000 usec,
   Tunnel transport MTU 1452 bytes

Router#sh ip interface Tunnel0 | include MTU
   MTU is 1452 bytes
```

```
Router#show interface tunnel 1 | include Path MTU
  Path MTU Discovery, ager 10 mins, min MTU 92
```

```
Router(config)#interface tunnel 1
Router(config-if)#tunnel path-mtu-discovery ?
  age-timer  Set PMTUD aging timer
  min-mtu    Min pmtud mtu allowed
  <cr>

Router(config-if)#tunnel path-mtu-discovery age-timer ?
  <10-30>   Aging time
  infinite  Disable pathmtu aging timer

Router(config-if)#tunnel path-mtu-discovery min-mtu ?
  <92-65535>  Bytes
```

```
Router(config)#crypto ikev2 profile default
Router(config-ikev2-profile)#authentication local eap ?
  gtc        eap method gtc credentials
  md5        eap method md5 credentials
  mschapv2   eap method mschapv2 credentials
  <cr>
Router(config-ikev2-profile)#authentication local eap md5 ?
  password   EAP password
  username   EAP username
```

```
Router(config-crypto-map)#set isakmp-profile IKEv1
Router(config-crypto-map)#set ikev2-profile IKEv2
% WARNING: Configuring Ikev2 profile will remove the Ikev1 profile configured
under this ipsec profile
```

```
Router#show crypto eli
Hardware Encryption : ACTIVE
 Number of crypto engines = 3

 CryptoEngine Onboard VPN details: state = Active
 Capability    : IPPCP, DES, 3DES, AES, GCM, GMAC, IPv6, GDOI, FAILCLOSE, HA

 CryptoEngine Software Crypto Engine details: state = Active
 Capability    : IPPCP, DES, 3DES, AES, SEAL, GCM, GMAC, RSA, IPv6, GDOI,
FAILCLOSE, HA

 IKE-Session   :     2 active,  2100 max, 0 failed
 IKEv2-Session :     2 active,  2100 max, 0 failed
 DH            :     1 active,  1050 max, 0 failed
 IPSec-Session :     0 active,  1000 max, 0 failed
```

```
Router(config)#crypto ipsec ?
  client                Configure a client
  df-bit                Handling of encapsulated DF bit.
  fragmentation         Handling of fragmentation of near-MTU sized packets
  ipv4-deny             Configure global ipv4 deny policy.
  nat-transparency      IPsec NAT transparency model
  optional              Enable optional encryption for IPSec
  profile               Configure an ipsec policy profile
  security-association  Security association parameters
  transform-set         Define transform and settings

Router(config)#crypto ipsec security-association ?
  dummy      Enable transmitting dummy packets
  ecn        Handling of ECN bit
  idle-time  Automatically delete IPSec SAs after a given idle period.
  lifetime   security association lifetime
  replay     Set replay checking
```

```
crypto isakmp key cisco address 10.10.10.2
```

```
interface Loopback0
 ip address 192.168.10.1 255.255.255.0
```

```
interface Tunnel0
 ip address 172.16.1.1 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.2
 tunnel protection ipsec profile ikev1
```

```
interface Ethernet0/0
 ip address 10.10.10.1 255.255.255.0
```

```
crypto isakmp policy 10
 authentication pre-share
crypto isakmp key cisco address 10.10.10.1

crypto ipsec profile ikev1

interface Loopback0
 ip address 192.168.20.1 255.255.255.0

interface Tunnel0
 ip address 172.16.1.2 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.1
 tunnel protection ipsec profile ikev1

interface Ethernet0/0
 ip address 10.10.10.2 255.255.255.0

router eigrp 1
 network 172.16.1.0 0.0.0.255
 network 192.168.20.0
```

```
Router1#show crypto isakmp sa
IPv4 Crypto ISAKMP SA
dst             src             state          conn-id status
10.10.10.2      10.10.10.1      QM_IDLE           1003 ACTIVE

IPv6 Crypto ISAKMP SA
```

```
Router1#show ip route 192.168.20.0
Routing entry for 192.168.20.0/24
  Known via "eigrp 1", distance 90, metric 27008000, type internal
  Redistributing via eigrp 1
  Last update from 172.16.1.2 on Tunnel0, 00:25:33 ago
  Routing Descriptor Blocks:
  * 172.16.1.2, from 172.16.1.2, 00:25:33 ago, via Tunnel0
      Route metric is 27008000, traffic share count is 1
      Total delay is 55000 microseconds, minimum bandwidth is 100 Kbit
      Reliability 255/255, minimum MTU 1476 bytes
      Loading 1/255, Hops 1
```

```
access-list 50 permit 192.168.10.0 0.0.0.255
```

```
crypto ikev2 authorization policy default
 route set interface
 route set access-list 50
 route accept any distance 100
```

```
crypto ikev2 profile ikev2
 match identity remote address 10.10.10.2 255.255.255.255
 authentication remote pre-share key cisco123
 authentication local pre-share key cisco123
 dpd 30 10 on-demand
 aaa authorization group psk list default default
```

```
interface Tunnel10
 ip address 172.16.2.1 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.2
 tunnel protection ipsec profile ikev2 ikev2-profile ikev2
```

```
access-list 50 permit 192.168.20.0 0.0.0.255

crypto ikev2 authorization policy default
 route set interface
 route set access-list 50
 route accept any distance 100

crypto ikev2 profile ikev2
 match identity remote address 10.10.10.1 255.255.255.255
 authentication remote pre-share key cisco123
 authentication local pre-share key cisco123
 dpd 30 10 on-demand
 aaa authorization group psk list default default

crypto ipsec profile ikev2
 set ikev2-profile ikev2

interface Tunnel10
 ip address 172.16.2.2 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination 10.10.10.1
 tunnel protection ipsec profile ikev2 ikev2-profile ikev2
```

```
Router1#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         10.10.10.1/500        10.10.10.2/500        none/none            READY
      Encr: AES-CBC, keysize: 256, PRF: SHA512, Hash: SHA512, DH Grp:5, Auth sign:
PSK, Auth verify: PSK
      Life/Active Time: 86400/629 sec
      CE id: 1004, Session-id: 2
      Status Description: Negotiation done
      Local spi: CDB7428C1D03F3AB       Remote spi: F5C4546346F689A9
      Local id: 10.10.10.1
      Remote id: 10.10.10.2
      Local req msg id:  3              Remote req msg id:  0
      Local next msg id: 3              Remote next msg id: 0
      Local req queued:  3              Remote req queued:  0
      Local window:      5              Remote window:      5
      DPD configured for 30 seconds, retry 10
      Fragmentation not configured.
      Extended Authentication not configured.
      NAT-T is not detected
      Cisco Trust Security SGT is disabled
      Initiator of SA : Yes
      Remote subnets:
      192.168.20.0 255.255.255.0
```

```
Router1#show ip route 192.168.20.0
Routing entry for 192.168.20.0/24
  Known via "eigrp 1", distance 90, metric 27008000, type internal
  Redistributing via eigrp 1
  Last update from 172.16.1.2 on Tunnel0, 00:11:56 ago
  Routing Descriptor Blocks:
  * 172.16.1.2, from 172.16.1.2, 00:11:56 ago, via Tunnel0
      Route metric is 27008000, traffic share count is 1
      Total delay is 55000 microseconds, minimum bandwidth is 100 Kbit
      Reliability 255/255, minimum MTU 1476 bytes
      Loading 1/255, Hops 1
```

```
Router1(config)#interface tunnel 0
Router1(config-if)#shutdown
is_up: Tunnel0 0 state: 6 sub state: 1 line: 0
 RT: interface Tunnel0 removed from routing table
 RT: del 172.16.1.0 via 0.0.0.0, connected metric [0/0]
 RT: delete subnet route to 172.16.1.0/24
 RT: del 172.16.1.1 via 0.0.0.0, connected metric [0/0]
 RT: delete subnet route to 172.16.1.1/32
 %DUAL-5-NBRCHANGE: EIGRP-IPv4 1: Neighbor 172.16.1.2 (Tunnel0) is down: interface
down
 RT: delete route to 192.168.20.0 via 172.16.1.2, eigrp metric [90/27008000]
 RT: no routes to 192.168.20.0, delayed flush
 RT: delete network route to 192.168.20.0/24
 RT: updating static 192.168.20.0/24 (0x0)  :
    via 0.0.0.0 Tu10  0 1048578
 RT: add 192.168.20.0/24 via 0.0.0.0, static metric [100/0]
```

```
Router1#show ip route 192.168.20.0
Routing entry for 192.168.20.0/24
  Known via "static", distance 100, metric 0 (connected)
  Tag 1
  Routing Descriptor Blocks:
  * directly connected, via Tunnel10, permanent
      Route metric is 0, traffic share count is 1
      Route tag 1
```

```
Router(config)#crypto ipsec profile ipsec
Router(ipsec-profile)#set isakmp-profile ikev1
Router(ipsec-profile)#set ikev2-profile ikev2
% WARNING: Configuring Ikev2 profile will remove the Ikev1 profile configured
under this ipsec profile!
```

```
Router(config)#interface tunnel 1
Router(config-if)#tunnel protection ipsec profile ipsec_profile ?
  ikev2-profile    Specify ikev2 profile for the crypto connection.
  isakmp-profile   Specify isakmp profile for the crypto connection.
  shared           Use a shared socket for the crypto connection.

Router(config-if)#tunnel protection ipsec profile ipsec_profile ikev2-profile
ikev2
```

```
Router#show crypto ipsec sa

interface: Virtual-Access1
    Crypto-map tag: Virtual-Access1-head-0, local addr 10.10.10.1

   protected vrf: (none)
   local  ident (addr/mask/prot/port): (10.10.10.1/255.255.255.255/47/0)
   remote ident (addr/mask/prot/port): (10.10.10.2/255.255.255.255/47/0)
   current_peer 10.10.10.2 port 500
     PERMIT, flags={origin_is_acl,}
    #pkts encaps: 726, #pkts encrypt: 726, #pkts digest: 726
    #pkts decaps: 1521, #pkts decrypt: 1521, #pkts verify: 1521
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0
    #pkts not decompressed: 0, #pkts decompress failed: 0
    #send errors 0, #recv errors 0

     local crypto endpt.: 10.10.10.1, remote crypto endpt.: 10.10.10.2
     plaintext mtu 1458, path mtu 1500, ip mtu 1500, ip mtu idb Ethernet0/0
     current outbound spi: 0x3F82A9C5(1065527749)
     PFS (Y/N): Y, DH group: group19
```

```
Router#show crypto ikev2 stats
--------------------------------------------------------------------------------
                           Crypto IKEv2 SA Statistics
--------------------------------------------------------------------------------
System Resource Limit:    0         Max IKEv2 SAs: 0         Max in nego(in/out):
40/400
Total incoming IKEv2 SA Count:     2         active:         2         negotiating: 0
Total outgoing IKEv2 SA Count:     0         active:         0         negotiating: 0
…

Router#show crypto session brief
Status: A- Active, U - Up, D - Down, I - Idle, S - Standby, N - Negotiating
        K - No IKE
ivrf = (none)
           Peer      I/F        Username          Group/Phase1_id   Uptime Status
     10.10.10.2      Vi1                  hostname=R2.cisco.com,cn 01:30:00    UA
     10.10.10.1      Vi2                  hostname=R1.cisco.com,cn 00:07:00    UA
```