A survey and analysis of TLS interception mechanisms and motivations

Exploring how end-to-end TLS is made "end-to-me" for web traffic

XAVIER de CARNÉ de CARNAVALET, School of Computer Science, Carleton University, Canada
PAUL C. van OORSCHOT, School of Computer Science, Carleton University, Canada

arXiv:2010.16388v1 [cs.CR] 30 Oct 2020

TLS is an end-to-end protocol designed to provide confidentiality and integrity guarantees that improve end-user security and privacy. While TLS helps defend against pervasive surveillance of intercepted unencrypted traffic, it also hinders several common beneficial operations typically performed by middleboxes on the network traffic. This issue has resulted in some parties proposing various methods that "bypass" the confidentiality goals of TLS by playing with keys and certificates essentially in a man-in-the-middle solution, and leads to new proposals that extend the protocol to accommodate third parties, delegation schemes to trusted middleboxes, and fine-grained control and verification mechanisms. To better understand the underlying motivation of such research proposals, we first review the use cases expecting plain HTTP traffic and discuss the extent to which TLS hinders these operations. We retain 19 scenarios where access to unencrypted traffic is still relevant and evaluate the incentives of the stakeholders involved. Second, we survey techniques and proposals by which TLS no longer delivers end-to-end security, and by which the notion of an "end" changes. We therefore include endpoint-side middleboxes and mid-path caching middleboxes such as Content Delivery Networks (CDNs), alike. Finally, we compare each scheme based on deployability and security characteristics, and evaluate their compatibility with the stakeholders' incentives. Our analysis leads to a number of findings and observations that we believe will be of interest to practitioners, policy makers and researchers.

1 INTRODUCTION

End-user security and privacy has improved due to the recent sharp increase in end-to-end TLS-encrypted web traffic [62], i.e., HTTPS. However, as the Internet traffic originally carried mostly plaintext communications, many network-related practices have been built relying on the fact that the plaintext data is readily accessible [77]. Unsurprisingly, and contrary to some stakeholders view, the shift towards HTTPS is reported to disturb a number of legitimate operations regularly performed by software or hardware middleboxes on plaintext network traffic, including [7, 14, 77]: network management and monitoring, problem troubleshooting, performance optimization, caching, intrusion detection, malware download and data leakage prevention, fraud monitoring, parental controls.

As a first solution that comes to mind, many client-facing security products and enterprise network gateways intercept HTTPS traffic by terminating and re-creating TLS sessions to access the plaintext [32, 38]. However, unlike with plain HTTP, any means of intercepting HTTPS traffic is seen as a threat by many stakeholders. This conflict was illustrated during the standardization of TLS 1.3, whereby some industry participants pushed for means to facilitate traffic inspection, while privacy and security advocates raised concerns about numerous weaknesses introduced by middleboxes in the past [44, 104]. In the end, TLS 1.3 was made less friendly to passive monitoring (by removing non forward-secret ciphersuites), resulting e.g., in the banking industry to promote as a competing standard an interception-friendly protocol: Enterprise TLS (ETS), opposed by, e.g., the EFF [56].

Industry actors have sought to minimize the deployment cost of interception by focusing on basic technical solutions compatible with TLS, e.g., sharing decryption keys (e.g., by CDNs below), trusting custom root certificates. Those solutions provide full access to the plaintext and little oversight, which severely puts at risk the benefits of TLS. Thus, even when a middlebox is allowed to intercept HTTPS traffic, it is desirable to limit/audit its privileges by providing restricted access (e.g., read but not write, with the URL path visible but not the full request or body),

Authors' address: Xavier de Carné de Carnavalet, email: xavier.decarnavalet@carleton.ca; Paul C. van Oorschot, School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, ON, K1S 5B6, Canada; email: paulv@scs.carleton.ca.

https://t.me/learningnets

or attesting the middlebox software. To this end, various groups have proposed or developed TLS variants to accommodate third parties.

Moreover, the emergence of Content Distribution Networks (CDNs) and the Internet of Things (IoTs) extends the notion of traffic interception and poses significant challenges to security and privacy. In particular, for CDNs to serve content over HTTPS, it is common for the content provider to either share their certificate private key with the CDN, or allow the CDN to obtain certificates for their domain (e.g., by pointing the domain name to the CDN itself) [71]. Therefore, CDNs change the notion of the "end" in end-to-*end* encrypted TLS sessions, pushing for new ways to represent an authorization or delegation to the CDN as a third party.

In this paper, before we jump into a technical analysis of proposals from the literature, we first lay down the necessary foundations by reviewing common network practices that rely on access to plaintext HTTP traffic, and the documented impact of TLS on them. We notably rely on RFC 8404 [77] to document the industry's perspective on the *pervasive encryption* of data at-rest and in-transit, and distill meaningful arguments with respect to TLS. We discuss the extent of the impact of TLS, especially TLS 1.3, and identify 19 actual use cases that benefit or require access to the unencrypted traffic. We also evaluate the incentives of the main stakeholders that support these use cases.

Then, we examine the techniques and proposals that aim at introducing a third party in the communication or enabling passive monitoring. Through a thorough literature review and our familiarity with the literature, we select for discussion the most prominent schemes, and also consider the most viable ones and those with highly interesting features. We thus review various ways by which long-term keys, session keys or intermediate secrets are shared, especially in the context of CDNs. We also explore the ways to achieve delegation with special-purpose certificates or tokens, and methods proposed to guarantee the secrecy of middlebox operations. Variants of TLS that handshake with middleboxes to achieve fine-grained control are also discussed. Finally, we provide a comparative evaluation of these schemes based on 27 security and deployment criteria. As an example of our findings, our analysis leads to the conclusion that generic middlebox-friendly solutions address no use case well.

The rest of the paper is organized as follows. We first describe regular network operations that expect access to plaintext HTTP (§3), and discuss the extent of the impact of TLS on them (§4). In §5, we explore the most common techniques employed by enterprises and client software to gain access to the plaintext of HTTPS traffic, along with prominent new proposals such as ETS. Caching middleboxes constitute a unique problem, for which we discuss related ideas and proposals in §6. Proposals that deeply change TLS to make it friendly to middleboxes are discussed in §7. Ways to reduce trust requirements on middleboxes are summarized in §8. Our systematization leads to the categorization and evaluation of the techniques and proposals against a 27-criteria framework in §9, with various insights in §10. Among main contributions are summaries provided in Table 3 and Table 4. The rest of the paper provides background and information to support and understand our analysis.

2 SSL/TLS: BACKGROUND

The reader familiar with TLS may choose to skip this section. We provide herein a review of important aspects of the SSL/TLS protocol, which are helpful to understand the rest of the paper. We focus on the latest major revisions only: versions 1.2 and 1.3, and we do not discuss exotic configurations. The interested reader is referred to related RFCs [34–36, 47, 95], books and publications [28, 94, 97], [15, Chapter 6] for more details.

2.1 TLS handshake overview

The TLS protocol performs an authenticated key exchange (or key transport) during an initial handshake to establish a shared secret from which symmetric encryption keys are derived. Prior to TLS 1.3, this key exchange is either based on RSA, Finite Field Diffie-Hellman (DH) or Elliptic Curve DH (ECDH). TLS 1.3 removed the RSA

https://t.me/learningnets

option. (EC)DH only uses ephemeral keys in 1.3 or may do so in prior versions, and is then referred as (EC)DHE. Ephemeral key exchanges provide forward secrecy, i.e., an attacker with the knowledge of the server's long-term key cannot decrypt past recorded sessions as the ephemeral private key has been deleted.

In the RSA key transport, a pre-master secret (PMS) is chosen by the client and encrypted using the server's RSA long-term public key. The server then decrypts the encrypted PMS (ePMS) and derives the shared master secret (MS) from the PMS and client/server-provided random nonces. With DH key exchanges, the server sends a public key share signed with its long-term private key. The client verifies the signature and combines the server's key share with its own private key share to derive the PMS. Conversely, the server combines its private key share with the client's public key share. Once the handshake is finished, the rest of the communication is encrypted with keys derived from the MS.

2.2 Protocol messages

We describe below the messages exchanged during the TLS handshake to achieve key exchange/transport.

2.2.1 ClientHello. The client begins the handshake by sending a ClientHello. Depending on the TLS version, the format of these messages slightly differs.

TLS 1.2. The ClientHello lists the maximum TLS version supported, a random number, a session ID when resuming a previous session, and a list of ciphersuites ordered by preference that specify cryptographic primitives that the client supports for key exchange, encryption, and message authentication. As mentioned above, the key exchange is either RSA, or (EC)DH(E). The encryption ciphers could be e.g., AES or ChaCha20. The mode of operation for block ciphers is also specified, and either provides authenticated encryption with additional data (AEAD, e.g., GCM and CCM) or not (e.g., CBC). Finally, the message authentication is performed using an HMAC based on e.g., SHA256 or SHA384.

The ClientHello also includes a list of TLS extensions. Notable ones include the server_name_indication (SNI) that specifies the server name (i.e., domain name); supported_groups with a list of supported elliptic curves for the (EC)DHE key exchange, e.g., SECP256r1, SECP384r1, x25519; and signature_algorithms with a list of supported signature algorithms used to authenticate the (EC)DHE parameters, e.g., RSA or ECDSA, with various padding and message digest algorithms.

TLS 1.3. The ClientHello's maximum supported TLS version field is set to 1.2 while all the supported versions are indicated in a separate TLS extension to avoid middlebox intolerance to the newer protocol. Similarly, the session ID is set to a random value to mimic a session resumption, while the feature is achieved through a separate Pre-Shared Key (PSK) TLS extension. The ciphersuites are different than in TLS 1.2 as each algorithm is selected separately. The ciphersuite no longer specifies the primitives for key exchange, which is forced to (EC)DHE. All the ciphersuites include AEAD ciphers. The ClientHello in TLS 1.3 also includes the client's key share based on its preferred group/EC.

2.2.2 ServerHello. The server selects the highest version it supports and that the client also advertised. It choses among the client's ciphersuites (respecting or not the client's preferred order), ECs, and signature algorithms. In TLS 1.3, if the server does not support the client's chosen EC, it sends a HelloRetryRequest with a list of the server's supported curves, and the client resends a ClientHello with a different key share.

The server then answers the client with a ServerHello that indicates the chosen parameters, a random value, and an optional session ID for the client to resume the session at a later time prior to TLS 1.3. The server also includes its key share in TLS 1.3, at which point both the client and the server can already derive a shared secret, called handshake_secret. Note that this key exchange is not yet authenticated.

2.2.3 Certificate. In TLS 1.2, the server continues by sending a Certificate message after the ServerHello with the server's certificate chain. The chain is composed at least of an end-entity certificate (EEC, also called a leaf

https://t.me/learningnets

certificate). Additional intermediate Certificate Authority (CA) certificates may be included for clients to establish a chain to a trusted root CA. In TLS 1.3, the Certificate and subsequent handshake messages are encrypted using keys derived from the handshake_secret. The (EC)DHE parameters are authenticated next by the CertificateVerify message that contains a hash of the handshake messages signed by the server's certificate private key.

2.2.4 KeyExchange. Prior to TLS 1.3, if the negotiated ciphersuite is not based on RSA, the server then sends its signed key share in the ServerKeyExchange message and ends with an empty ServerHelloDone message. The client responds with its key share in a ClientKeyExchange message, wasting round trips compared to TLS 1.3. For RSA ciphersuites, the ServerKeyExchange is skipped, and the client simply sends the ePMS in the ClientKeyExchange message.

2.2.5 Finish. Finally, both the server and then the client send to each other a MAC of all the handshake messages, encrypted with the appropriate derived keys as described below.

2.3 Channel encryption keys

Once a MS has been established, the two ends derive a number of keys. The client and server encrypt their messages with a separate set of symmetric keys, MAC keys and Initialization Vectors (IVs), respectively called client_write_{key,MAC_key,IV} and server_write_{key,MAC_key,IV}. When using AEAD ciphers, the {client,server}_write_key serve as the MAC keys.

The derivation of these keys from the MS is based on a Pseudorandom Function prior to TLS 1.3 as defined in the TLS standard, while version 1.3 leverages another construction based on the HKDF defined in RFC5869 [63].

In TLS 1.3, a shared secret, handshake_secret, is already established early during the handshake. Four handshake keys and IVs are derived to encrypt the rest of the handshake messages, i.e., {client,server}_handshake_{key,iv}, in a similar fashion to TLS 1.2 encryption keys. The MS is also derived from handshake_secret, and leads to separate application keys and IVs, i.e., {client,server}_application_{key,iv}, which in turn encrypt the rest of the communication.

A summary of the dependence between secrets and keys in TLS 1.2 and 1.3 is provided in Figures 1 and 2, respectively.

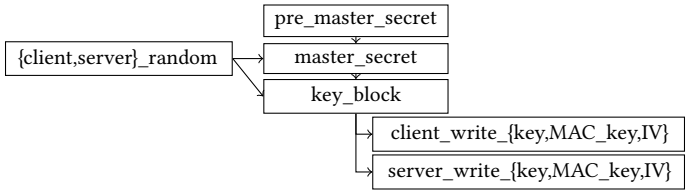

Fig. 1. Dependence between secrets and keys in TLS 1.2

2.4 Certificate validation

When validating the server certificate, the client tries to chain the certificates it received to a root CA certificate it already trusts, by verifying the signatures. The list of trusted CAs is typically preloaded with the operating system or individual browsers and updated periodically. Then, the client checks whether the leaf certificate is appropriate for the contacted domain for web purposes, and that all certificates in the chain are not expired or revoked. Certain browsers also mandate that the certificates be recorded in multiple Certificate Transparency (CT) logs. This is verified by the presence of a Precertificate Signed Certificate Timestamp (SCT) issued by the

https://t.me/learningnets

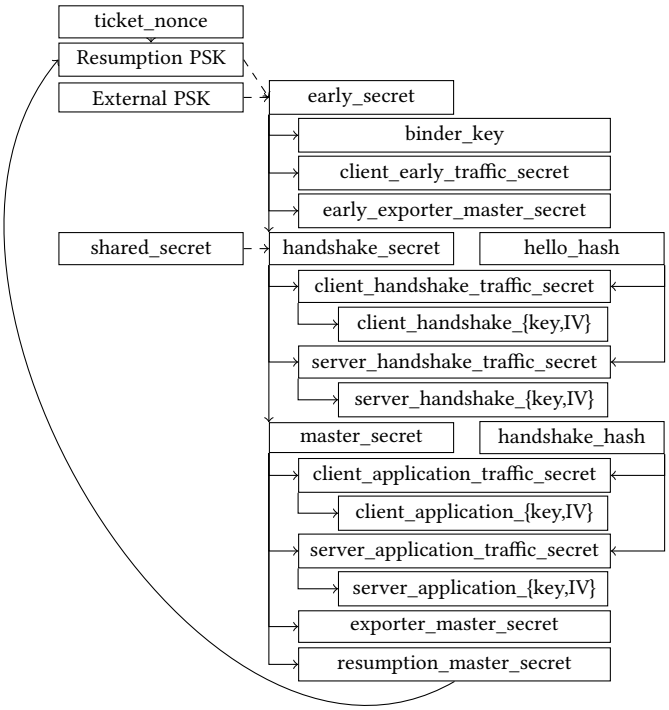

Fig. 2. Dependence between secrets and keys in TLS 1.3

logs. The SCT is either embedded in the final certificate or provided to the client through a TLS extension. Finally, the client may require that only specific leaf or issuing certificates be used, by pinning their keys (cf. HPKP [42]).

3 OPERATIONS EXPECTING HTTP ACCESS

This section builds the foundations for the rest of the paper by listing the common network practices that rely on access to plain HTTP traffic. These practices are positioned as the reason that TLS interception is needed. We identify four main categories of use cases: legal, security, performance-related and other reasons. Use cases are negatively impacted or precluded by the deployment of HTTPS. An open question for consideration is: do these practices (use cases) provide sufficient justification for HTTPS interception, or is interception simply the easiest way to accommodate the existing practices? This actual impact of TLS on these practices is discussed in §4. We group middleboxes operating on the traffic based on their location on the network path between the endpoints, i.e., client-side (up to the user's ISP), mid-path (including CDNs, although edge servers are usually located closer to the user), and server-side. Figure 3 illustrates the location of these middleboxes on the network.

3.1 Government-driven "legal" common practices for HTTP access

We describe below legal requirements for processing cleartext network traffic as it pertains to web traffic.

Lawful interception. Lawful interception is a constraint that a government can impose on operators to obtain a specific portion of the traffic they are carrying, typically upon delivering a warrant or subpoena. Specific architectures and protocols for lawfully intercepting IP traffic have been standardized [9, 39]. Note that this

https://t.me/learningnets

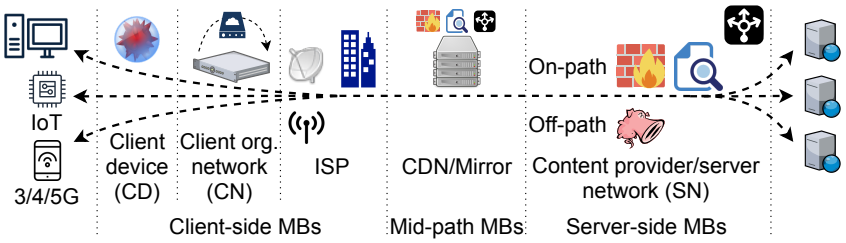

Fig. 3. Location of middleboxes (MB) impacted by TLS on the connection path

practice is distinct from warrantless surveillance (cf. Snowden's revelations [53]), which we do not consider here as a use case.

ISP data retention. Internet service providers (ISP) may be legally required to keep network traffic records to comply with data retention laws. Those laws vary per country. For instance, in the European Union (EU), the Data Retention Directive was adopted in 2006 and specified the retention from six months to two years of various pieces of metadata. This directive was invalidated in 2014 by the Court of Justice of the EU [31]; however, numerous EU countries continue to apply their own retention laws [43]. In Australia, an equivalent data retention law has passed in 2015 and is still effective as of 2020 [8]. In the United States, there is no law to our knowledge that mandates ISPs to retain data.

Content filtering mandated by governments. A government may impose content filtering policies on network operators [77]. The policies could be targeted at blocking access to illegal websites, related to e.g., child abuse, content piracy.[1] Content filtering may be implemented at various layers [59]: 1) blocking access to certain IP addresses or to certain protocols; 2) blocking DNS resolution to targeted domains; 3) recognizing content to be blocked by keyword matching or fingerprinting from a blacklist; 4) blocking known URLs; and 5) unlisting the websites from search engines. Only #3 and #4 assume access to cleartext web traffic.

Fraud detection. Certain industries perform fraud monitoring (e.g., stock fraud) of certain employees as part of out-of-band traffic inspection [14, 58].

3.2 Security-based common practices for HTTP access

We list below some prominent examples of security-related practices that also rely on cleartext traffic.

Malware/intrusion detection/prevention. Network security functions, e.g., Intrusion Detection/Prevention Systems (IDS/IPS) and Web Application Firewalls (WAF), control network communications in search for attack patterns or to detect vulnerabilities [7, 14]. They require access to the application-layer traffic. Also, antivirus (AV) solutions control the traffic to prevent malware downloads and to block known malicious URLs [7, 14].

Data retention for compliance. Organizations and data center operators may also want to keep network packet captures to investigate attacks (forensics), or demonstrate compliance to industry standards [77]. Note that such industry standards may not be government-driven and are not enforced by law.

DDoS mitigation. CDNs may also provide, beyond simply caching content, volumetric Distributed Denial-of-Service (DDoS) attack protection. To enable access to users who were incorrectly flagged as attackers, CDNs sometimes serve a CAPTCHA challenge before the request is honored to differentiate between human and automated queries (also called "automatic bot discernment"). This mechanism is also used to block anonymous

[1] See the blocking of The Pirate Bay. https://en.wikipedia.org/wiki/Countries_blocking_access_to_The_Pirate_Bay

https://t.me/learningnets

queries to reach a website, e.g., when using Tor (see [61]). Application-layer (L7) DDoS attacks are more difficult to automatically detect than volumetric DDoS attacks. The former requires inspection of the application-layer traffic to understand that e.g., a resource-hungry feature of a website is being abnormally triggered.

Data loss prevention. Companies may want to monitor outgoing traffic to stop data exfiltration and prevent information loss [7, 14, 60]. These tools try to identify (possibly obfuscated) sensitive data in the traffic.

3.3 Performance-based common practices for HTTP access

There are several optimization mechanisms that rely on information about the traffic to improve performance-related decisions, ranging from load balancing and Quality of Service (QoS) adaptation, to data broadcasting and caching.

Application load balancing. Network loads can be balanced in a number of ways to back-end servers. A fine-grained approach consists in making decisions at the application layer, e.g., based on the request path (cf. AWS Application Load Balancer (ALB) [4]). According to RFC8404 [77], cellular networks may route traffic depending on application-layer criteria. This could be the case e.g., as part of the Mobility Load Balancing (MLB) function of 3GPP Self-Organizing Networks (SONs) where congested cells can hand over some load to other less busy cells. Another cellular network optimization consists in queuing traffic based on its bandwidth and latency needs. To accurately detect the type of application traffic, signatures are matched by a monitoring system that performs deep-packet inspection (DPI). Finally, for video streaming media, a dynamic QoS can be adapted based on the video bit-rate information obtained from plaintext HTTP requests (e.g., as done in [81]).

Broadcasting. Content broadcasting helps reduce bandwidth requirements by sending the data once for a given region, which is then locally redistributed to all clients. This is especially useful in satellite communications where a satellite can broadcast the signal to many receivers [46]. Similarly, the 3GPP Multimedia Broadcast/Multicast Services (MBMS) specification aims at distributing live TV content to trusted edge proxies, to be delivered to the final users through either unicast or multicast.

Caching and replication. Caching popular content near the network edge is a way to reduce unnecessary bandwidth consumption and latency. Content caching and replication may be performed upon agreement between the content service provider and CDNs, whereby the former delegates its content to CDNs. Caching may also be transparent to the end parties, when performed e.g., by ISPs [77, 83], although this practice is controversial [77].

Transcoding and compression. Mobile devices with slow connections, limited data caps, short battery life or slow CPUs may benefit from transparent proxies that (re)compress web contents in an effort to minimize transfer and processing times [77]. This service can be built into browsers, e.g., Opera Mini [89], Google's Flywheel Chrome extension [1]. Such performance-enhancing proxies can downsize image resolution, video and audio quality [83, 114]. Mobile network operators have been shown to compress images [24]. Also, performance-enhancing proxies at the network edge, as used on mobile or satellite links, could "further control pacing, limit simultaneous HD videos, or prioritize active videos against new videos" [77].

3.4 Other common practices motivating HTTP access

Here we list other cases that do not fit into the previous three categories.

Diagnosing and troubleshooting issues. Network packet captures traditionally provide visibility into networks, assist in troubleshooting problems, and are used to analyze application performance [52, 77]. Captures can even be provided by customers as taken close to the client-side, allowing for a targeted and efficient troubleshooting. The ability to observe actual network communications is often the easiest, when not the last, resort to troubleshoot a problem.

https://t.me/learningnets

Traffic filtering. Parental control solutions also filter network traffic to block forbidden contents, remove explicit images, sensor swear words [32]. Similarly, ad blockers may choose to remove unwanted ads from the web traffic before it reaches the browser. Finally, employers may want to regulate the use of their resources by employees by, e.g., collecting browsing histories, preventing abusive use of bandwidth [7]. Similar filtering needs to legal constraints on ISPs may apply.

Content-based billing. Network operators may leverage fine-grained billing of their customers based on the website or URL visited to, e.g., provide free-of-charge access certain contents (zero-rating [22]), or charge more for premium services [27].

Controversial/abusive practices. ISPs may insert tracking headers [16, 107], with or without personally identifiable information such as a phone's IMEI and IMSI. Sometimes, this is simply intended to pass information between internal parts of the operator's network; however, this may also be done for advertisement and tracking purposes [27, 77, 107]. ISPs could also insert interstitials to inform users of e.g., quota limits [77]. Finally, they may also monetize HTTP errors by inserting ads on error pages [114]. We do not further consider these controversial practices by ISPs in this study.

The impact of the above-mentioned cases is discussed next.

4 CHALLENGES INTRODUCED BY TLS

We discuss in this section the impact of TLS on the aforementioned use cases (§3), as reported in IETF and industry discussions [7, 14, 77], and various proposals [58, 83]. We hint at obvious alternative solutions when such use cases can be achieved without impeding TLS. We also observed that regulations are often cited throughout these references as a major use case. We address this particular concern separately. Finally, we identify 19 use cases for access to HTTP traffic in the presence of TLS, and we perform an incentive analysis from the viewpoint of the main stakeholders.

4.1 Evaluation of HTTP access use cases in presence of TLS

We proceed by grouping use cases for HTTP access faced with similar challenges. Table 1 summarizes 19 use cases and evaluates the impact of HTTPS.

(1) *Lawful interception, data retention by ISPs.* While TLS does not prevent IP traffic interception, it diminishes its meaning. Protocol metadata, e.g., source and destination IP addresses, payload size, and access frequencies, are still available; however, application content is not. Lawful interception of selected individuals' traffic will only yield metadata for such websites that are reached over TLS. The ISP data retention laws we looked at (in the EU and Australia) seem to only aim at listing visited websites (visible in DNS queries and the TLS SNI extension) or simply target flows metadata, and therefore are not impacted by the switch from HTTP to HTTPS.

(2) *Content filtering and modification, broadcasting.* Fine-grained inspection and techniques that require access to the web request and response content are entirely defeated by TLS. This includes content filtering (including by ISPs), IDS/IPS, AV, DLP, fraud detection, parental controls, ad blockers, transcoding and compression. Instead, the inspection may be moved closer to the client when possible, as with browser add-ons commonly installed by AVs and for ad-blocking. Similarly, end-to-end encrypted traffic cannot simply benefit from broadcasting; however, see [46].

When the IPS/WAF systems are given the server's certificate private key (see details in §5.2), they are able to decrypt TLS traffic up to version 1.2. In TLS 1.3, as all ciphersuites support PFS, the long-term private key is insufficient to inspect the traffic.

We make a distinction between IDS/IPS placed on the client network to monitor outbound connections to external servers, and IDS/IPS/WAF on the CDN- or server-side that analyze inbound connections.

https://t.me/learningnets

(3) *DDoS.* For L7 DDoS protections, other application-agnostic markers such as traffic provenance or uncommon traffic patterns are still relevant indicators; however, effectively distinguishing attackers from legitimate traffic benefits from the access to the cleartext traffic [77]. This is achieved as part of IPS/WAF solutions discussed above.

Volumetric DDoS protection is agnostic to TLS, and therefore does not constitute a use case for access to the HTTP traffic. However, the pre-request authorization mechanism it is often associated with requires the ability to answer the client's request with a challenge prior to involving the end server.

(4) *Load balancing and QoS.* Assertions are often made that TLS impacts load balancing, and proposals try to address this problem [7, 64, 77, 83]. However, in general, load balancing is not dependent on application-layer information. Typical network load balancing deals with Layer 3 (IP) or 4 (TCP). When load balancers also act as TLS terminators, the load of decrypting TLS traffic is therefore not the bottleneck (it is not yet load balanced), and hence TLS does not introduce new challenges with such load balancers.

Only application-layer load balancers (ALBs) that are not TLS terminators suffer from encrypted TLS traffic. This seems to be the case in cellular networks when MLB, traffic queuing, or dynamic QoS is used [77]. In those three cases, TLS traffic prevents these functions from obtaining the relevant information to make decisions, possibly leading to a degraded user experience.

AWS ALB enables balancing based on the request host/path and query parameters; however, if the full requested URL is unavailable to the balancer, the application could be adapted in a way that balancing-related information appears in an outer layer, e.g., DNS or IP. The TLS SNI extension could also be used as a replacement for host-based routing.

ALBs typically communicate the source IP address to the next hop by inserting it as an HTTP header (e.g., X-Forwarded-For) when needed. Without terminating TLS sessions, such load balancers could simply encapsulate the traffic with, e.g., the PROXY protocol [103] that wraps TLS traffic with additional headers.

(5) *Tracking.* Content-based billing is inherently hindered by TLS. Tracking of users by operators for technical reasons is equally impacted; however, such tracking does not necessarily need access to the traffic content, therefore there could be other ways of carrying the metadata information, see the above item.

(6) *Caching.* When the content is encrypted, CDNs are unable to cache it. A possible delegation of unencrypted content breaks confidentiality assumptions and brings new trust issues. Other opportunistic caches done by e.g., ISPs, are hindered as well.

(7) *Problem resolution.* When troubleshooting difficult [14] or application-layer [77] problems, the opaque encrypted traffic leads to inaccuracies in the diagnosis process due to the inability to locate specific transactions, user identifiers, session identifiers, URLs, and time stamps [58]. This reduces the efficiency of repair services, which impacts the service availability as promised in Service Level Agreements (SLAs), and in turn incurs costs. RFC8404 [77] notes that in the absence of packet captures, application-provided diagnosis information is usually poor and unhelpful, which motivates the need to access the cleartext data.

In TLS 1.3 and TLS 1.2 with PFS ciphersuites, simply sharing a server's private key (see §5.2) with troubleshooting operators does not enable them to decrypt the traffic. For short troubleshooting times at the server side, all ephemeral keys could be exported (similar to §5.4.1).

When users suffer from an individual problem that requires targeted troubleshooting, they may be willing to participate in the effort by, e.g., running with modified configurations that are more friendly to debugging.

(8) *New use case: monitoring IoT devices.* The Internet-of-Things (IoT) is a concept whereby previously unconnected objects become network-aware, are remotely controlled and interact with online services. This trend is sufficiently recent for IoT devices to start encrypting their communications from day one, unlike most other cases presented above. The rise of consumer IoT products with multiple sensors within the user's private space sparked privacy violation concerns [93]. Therefore, end users may be interested in monitoring the traffic of their IoT

https://t.me/learningnets

devices for privacy reasons. IoT traffic monitoring is a novel area that faces similar challenges as in 2). However, the main difference is that the client is no longer a browser or application of the user's choice running on a general-purpose machine, but rather a vendor-baked client on the embedded device. Furthermore, IoT devices bring a unique challenge for traffic inspection as they sometimes make use of TLS client authentication using a (hardcoded) per-device certificate [18].

Table 1. Summary of use cases (motivations) for access to plaintext traffic in the presence of TLS (from §4.1), the impact of HTTPS in general on them and of TLS 1.3 in particular

	Use cases	Can be conducted with HTTPS traffic	Effectiveness with HTTPS (vs. HTTP traffic)	Impact of TLS 1.3 (vs. TLS 1.2)	Related operations	Immediate alternatives
Legal	*a.* Lawful interception	Yes, limited	Metadata only	Low†	–	–
	b. ISP data retention	Yes, limited	Limited browsing history	None	–	–
	c. Legal content filtering	Yes, limited	Less granularity	None	–	DNS filtering, IP blocking
	d. Fraud detection	No	None	None	Employee monitoring	Endpoint monitoring
Security/privacy	*e.* Malware download prevention	Yes, limited	Domain blocking only	None	–	Browser addon
	f. Outbound IDS/IPS	No	None	None	DLP (see *j*)	Endpoint monitoring
	g. Inbound IDS/IPS/WAF	No	None	No key sharing	L7 DDoS prevention, DLP (see *j*)	Earlier TLS termination
	h. Data retention (compliance)	Yes, limited	Metadata only	Low†	Attack investigation, forensics	Endpoint monitoring, earlier TLS termination
	i. Pre-request authorization	No	None	None	L7 DDoS prevention, Turing test	–
	j. Data loss prevention (DLP)	Yes, limited	Metadata only	None	IPS (see *f* and *g*)	Endpoint/server monitoring
	k. IoT device monitoring	Yes, limited	Metadata only	None	–	–
Performance	*l.* L7 load-balancing	No	None	None	–	–
	m. Broadcasting	No	None	None	–	–
	n. Server-mandated caching	No	None	None	CDN	–
	o. Opportunistic caching	No	None	None	ISP caching	–
	p. Transcoding/compression	No	None	None	–	–
Other	*q.* Problem troubleshooting *q1.* with user help *q2.* with no user help	Yes, limited	Metadata only	No key sharing	–	Endpoint monitoring, application logging
	r. Parental control	Yes, limited	Less granularity	None	–	Browser addon, special browser
	s. ISP billing/tracking	No	None	None	–	–

Red color denotes a negative impact for the third parties that want plaintext access. Note that the second column does not consider certificate private key sharing or TLS session splitting as a way to conduct operations, although they are common technical practices in some cases. However, we consider the impact of TLS 1.3 on such practices (fourth column). In the third column, metadata refers to e.g., domain information, traffic patterns, payload size. †Low impact: the third party does not learn the server certificate, and connections are forward-secure (which is not always the case with TLS 1.2).

https://t.me/learningnets

4.2 Industry regulations

Industry regulations may seem to require access to plaintext network traffic or prohibit its inspection, depending on the objective. We discuss the limited arguments stating that some regulations require access to plaintext traffic and would be negatively impacted by TLS.

An Internet Draft by Cisco [7] argues that industry regulations such as NERC Critical Infrastructure Protection (CIP) and the Payment Card Industry Data Security Standard (PCI-DSS) are impacted by TLS 1.3. However, we are unable to find documented evidence to support this claim. For instance, the authors point that a PCI-DSS requirement "*describes the need to be able to detect protocol and protocol usage correctness.*" We could not locate this specific requirement in the referenced version of the standard. The closest match would be Requirement 4.1.c which states: "*Select and observe a sample of inbound and outbound transmissions as they occur (for example, by observing system processes or network traffic) to verify that all cardholder data is encrypted with strong cryptography during transit.*" The stated verification could be achieved by asserting that: 1) no cardholder data is sent unencrypted, and that 2) when encrypted connections are detected, the negotiated ciphersuite is compliant, which is not hindered by TLS 1.3.

The draft also points to Requirement 10 about "*the need to provide network-based audit to ensure that the protocols and configurations are properly used*" [7]. We are unable to find support for this claim from reading the corresponding PCI-DSS standard requirement, and were unable to contact the draft's authors for clarification.

Similarly, we could not identify requirements in NERC CIP impacted by TLS, especially version 1.3. Overall, to our knowledge, there is no major industry standards that are affected by HTTPS compared to plain HTTP and specifically by TLS 1.3. If such conflicts actually exist, we urge industry regulators to document the problem to enable the community to find solutions.

4.3 Use cases vs. stakeholders

Considering the use cases that expect access to HTTP traffic described in §3 and the actual impact of TLS on them explained in §4.1, we summarize 19 derived use cases (listed from *a* to *s*) for access to HTTP traffic in the presence of TLS, which remain unaddressed by alternative solutions (see Table 2). Notable business use cases can be mapped to several of our use cases, e.g., CDNs could span over *g*, *i*, *l*, *n* and *p*.

Furthermore, since the access by a third party to the plaintext content of TLS communications is normally considered an attack on the protocol, at least one TLS endpoint needs to collude with the third party. That is, without the collaboration of a TLS endpoint (therefore of the end-user, its client or the server), these use cases are prohibited. We explore below the various incentives of multiple stakeholders for this collaboration, and list these incentives with respect to the 19 use cases in Table 2. We consider the stakeholders' incentives while evaluating proposals in §9.

Examples of incentives. These use cases are associated with different stakeholders with various degrees of interest. For instance, the end-user is unrelated by data retention compliance requirements faced by the server operator, and could even be adverse to their ISP inserting tracking headers. Likewise, the server entities are unlikely interested in how companies prevent data leakages from their network. However, several parties may have an interest to detect malware downloads: the end-user or enterprise network administrators in the first place for security reasons; the ISP could also gain business advantages by proposing a "more secure" Internet access to its users; finally, the server operators could choose to be compliant with solutions that assert their traffic is benign to enhance their public image.

Stakeholders. For the sake of our analysis, we consider five main stakeholders: the end-user (EU), client vendor/author (CV, e.g., browser vendor), end-user organization (EUO), ISP, CDN, and content provider (CP). However, note that the EUO may also outsource security operations on their traffic to cloud-based companies (see [64]). In this case, the organization may want to enable the cloud-based solution to inspect TLS traffic, yet

https://t.me/learningnets

Table 2. Incentives and requirements for the main stakeholders in each use case for access to plaintext traffic in the presence of TLS summarized from §4.1.

	Use cases \ Stakeholders	† End-user	† Client vendor	End-user organization	ISP	CDN	Content provider
Legal	*a.* Lawful interception	★			L		
	b. ISP data retention	★			L		
	c. Legal content filtering	★			L		
	d. Fraud detection	e★		LE			
Security/privacy	*e.* Malware download prevention	SL	S	SLE	e	e	e★
	f. Outbound IDS/IPS	sl★	s★	SLE			★
	g. Inbound IDS/IPS/WAF					SE	SLE★
	h. Data retention (compliance)			E			E
	i. Pre-request authorization	★				SE	SE
	j. Data loss prevention	★	★	SLE			SLE
	k. IoT device monitoring	S	se★				se
Performance	*l.* L7 load-balancing	f			FE	FE	FE
	m. Broadcasting	f			FE	e	e
	n. Server-mandated caching	f	f			FE	FE
	o. Opportunistic caching			FE	FE		
	p. Transcoding/compression	f★			FE	FE	f
Other	*q.* Problem troubleshooting	f					FE
	r. Parental control	S★	S		E		
	s. ISP billing/tracking	★			FE		

Categories of stakeholder incentives and requirements for access to plaintext traffic: legal (L), security and/or privacy (S), performance (F), and economic/business (E). We list the primary incentives/requirements under each stakeholder using capital letters (L,S,F,E), and denote by the corresponding lowercase letters (l,s,f,e) when a stakeholder could reasonably offer some help (if not too costly) to support the preceding reasons. A star indicates stakeholders that are opposed to plaintext access for that use case (★). † denotes an assumption of a non-malicious stakeholder.

may be reluctant to fully trust the provider not to take advantage of the unencrypted data. This situation sparks a separate problem in which the EUO is split into two entities.

Categories of incentives. We identify four categories of stakeholder incentives and requirements for access to plaintext traffic: legal, security and/or privacy, performance, and economic/business. For each use case for access to plaintext traffic in the presence of TLS and for each stakeholder, we list in Table 2 the corresponding incentives/requirements. A stakeholder may not always represent entities that share the same incentives. For instance, the author of a piece of malware acting as a TLS client may not willingly support functions that would defeat its intended purposes, the way browser vendors would. Similarly, the corresponding attacker-controlled server will not support features that would, e.g., facilitate intrusion detection in the victim's network.

Influence of stakeholders. Note that the EUO may be able to impose its incentives onto the user browsers and EU as seen in organizations that control user devices and enforce strict application whitelisting. End-users can also influence browser vendors to support a feature they want.

https://t.me/learningnets

No mutual agreement. Both the EU/CV/EUO and the CP never share a strong incentive to support a use case among the 19 considered. Note that for data retention for compliance (*h*) and data loss prevention (*j*), the incentives of both sides are disjoint. This observation is particularly relevant when designing solutions. Indeed, a proposal that requires the collaboration of both ends is unlikely to be supported.

5 SESSION SPLITTING AND KEY SHARING

Faced with immediate road blocks imposed by TLS, network operators' first recourse are simple solutions that bypass TLS by splitting TLS sessions, or sharing TLS keys/secrets once established. These solutions only require the middlebox and a single relevant endpoint to be compatible, while the other endpoint remains largely unaffected. In this section and the following ones, we list prominent issues for each technique and proposal discussed.

5.1 TLS session splitting, a.k.a. man-in-the-middle

The simplest solution to access the unencrypted traffic of an outbound connection is for a middlebox to interpose the TLS connection and *split* it into two sessions [60] (not to be confused with the encryption/authentication splitting described in [68]). This behavior is often (abusively) referred to as a man-in-the-middle attack. The client is directed to the middlebox acting as a server, which in turn acts as a client and exchanges with the intended server. Hence, the middlebox has two sides: client-side and server-side, respectively. The middlebox also acts as a CA with its own root certificate since it cannot simply impersonate the server using publicly trusted certificates. This method requires proper configuration of the client to accept middlebox-issued certificates. The middlebox is trusted to perform the client validations (e.g., certificate validation, minimum key length, strong signature hashing algorithm), and to maintain an acceptable level of security on the server-side connection (e.g., same TLS version and ciphersuites). This method is often adopted by antivirus, parental control, malware, student/employee monitoring systems, anti-ads software, and enterprise network appliances [32, 33, 38, 60, 111].

When the middlebox leverages a publicly trusted intermediate CA certificate, clients do not require the extra provisioning of the middlebox' root certificate. However, clients are mostly unaware that the interception is taking place, and OS/browser vendors sanctioned the issuing CAs when this deceptive practice was discovered (see [50, 69, 78, 79]). Therefore, splitting the TLS session is only relevant when considering a home or enterprise/university network.

Issues. The client does not see the server's certificate; thus this simple technique precludes any certificate-based verifications and enforcements by the client, e.g., certificate validation, revocation checking, key pinning [42], DANE [37]. Instead, the middlebox is trusted to perform such operations consistently with the client's expectations. Also, the server's use of EV certificates is downgraded to regular DV certificates; however, note that EV certificates for the web are no longer differentiated by major browsers [57, 86].

In practice, TLS session splitting middleboxes have been shown to overlook critical steps as well as introduce additional attack surface [32, 38, 90, 111]. This approach is also incompatible with TLS client authentication, unless the middlebox authenticates on behalf of the client to the server.

Finally, it is not always easy or even possible to provision clients to trust a custom root CA. Specifically, user devices that are not controlled by the organization that performs the session splitting needs to be provisioned by the user. Certain clients such as antivirus applications or IoT devices rely on their own trust store with little support for customization [32].

ProxyInfoExtension. An Internet draft [72] proposes that the middlebox carries the server's certificate back to the client in a new informational TLS extension, named ProxyInfoExtension, as part of the ServerHello to the client. The client is then responsible to perform validity checks and decide to continue or not the connection. The middlebox is trusted to place the actual server certificate in this extension, as there is no cryptographic binding between

https://t.me/learningnets

this certificate and the rest of the server-facing connection. Although incomplete to address all criticisms, this technique aims to take the middlebox out of certificate-related decisions, reducing potential security issues and bringing more transparency to the client.

5.2 Certificate private key sharing

When in possession of the server certificate's corresponding private key, two options arise.

5.2.1 Passive decryption using RSA certificates. When TLS endpoints negotiate an RSA key exchange ciphersuite, and thus the server presents an RSA certificate, the ePMS can be decrypted by any entity that possesses the certificate private key. Therefore, sharing the server's certificate private key is an effective solution to passively monitor recorded TLS traffic by, e.g., IDS [52, 58, 113].

Issues. This technique for passive monitoring is incompatible with DH key exchange ciphersuites, including TLS 1.2 PFS ciphersuites and TLS 1.3 altogether.

5.2.2 Active impersonation. The private key of an RSA or EC certificate can be shared for active inspection or impersonation of the server (e.g., content caching) as seen with several commercial CDNs [71]. This technique falls back to TLS session splitting (§5.1), but using the server certificate.

Issues. This technique and the previous one question the notion of "private" key that is no longer known to one entity only. More importantly, impersonating the final server (e.g., on a TLS-terminating CDN node, i.e., edge node) also challenges the notion of end-to-end encryption when the server-end may not be the actual end, and without the knowledge of the end-user.

5.3 Sharing DH key exchange private shares

We first describe the technique of sharing DH secrets, then discuss the ETS standard and the tls_visibility Internet draft.

5.3.1 Static DH key sharing. With (EC)DHE key exchanges, the session secret is only known to the two entities that know a private part of the key. However, unlike in the previous technique, new (EC)DHE keys are generated for each new session on both ends (i.e., they are ephemeral keys), which precludes pre-sharing a single key. Therefore, to facilitate traffic monitoring at the server-side, a simple solution is to make the server's DH key share static, then share the private part with a middlebox just as one would share an RSA certificate private key (§5.2). The TLS session secrets are reconstructed by the middlebox in the same way as the server does. Since RSA key exchange and static DH ciphersuites were dropped in TLS 1.3, this method comes as an attractive solution to re-enable passive monitoring in a way that is similar to the well-known certificate private key sharing.

Issues. The resulting semi-static DH key exchange no longer provides forward secrecy. Similar to §5.2, a private key share is no longer private to the end entity. However, server impersonation requires the active participation of the server to sign the CertificateVerify message. Therefore in this case, other alternatives are more suitable (see Keyless SSL 6.2.1). Moreover, if the client is unaware that the server uses a static key and also decides to choose a static key for its own needs (e.g., client-side passive monitoring), the resulting shared secret remains the same across sessions. The uniqueness of the derived TLS secrets then only depends on the client and server random values. If those values suffer from implementation flaws, plaintext communications may even be recovered [106].

5.3.2 Enterprise Transport Security. ETS [41] (formerly known as eTLS) is a variant of TLS 1.3, promoted by the Bank Policy Institute and standardized by the ETSI. It combines TLS splitting, server impersonation, and static DH keys on middleboxes/firewalls or enterprise servers to passively decrypt traffic within an enterprise. ETS addresses three scenarios:

https://t.me/learningnets

Scenario 1. Both clients and servers reside in the enterprise. The servers share static DH keys with passive middleboxes. Client-server connections do not benefit from PFS.

Scenario 2. The clients are located outside the enterprise network. The clients' TLS connections are first terminated at a firewall that impersonates the server as described in §5.2.2, then are re-encrypted within the enterprise up to the final server that employs a static DH key. This scenario is similar to the one described in an Internet Draft [52] in the context of monitoring re-encrypted traffic within datacenters, after the client-originating TLS traffic is terminated at a load balancer. Note that PFS is preserved on the client-firewall or client-datacenter segment using regular TLS stacks.

Scenario 3. Enterprise clients try to reach servers outside the enterprise network. A firewall at the edge of the network impersonates the end servers by splitting TLS sessions as in §5.1, albeit with static DH keys for other middleboxes to decrypt the traffic.

Criticisms. ETS received vivid criticisms due to its weakening of TLS 1.3. It has been assigned CVE-2019-9191 for lacking forward secrecy. An Internet draft by the American Civil Liberties Union (ACLU) [49] also proposes that clients reject connections to servers/middleboxes implementing ETS, while the Electronic Frontier Foundation (EFF) calls for ETS not to be used [56]. A request from ETSI to NIST aiming at delaying the publication of TLS configuration guidelines to include the near-finalized ETS standard was dismissed [98].

We note that in the three scenarios described in the ETS standard, TLS clients outside the enterprise never reach an ETS server directly. Rather, those client connections end at the edge firewall/load balancer that acts as the final TLS server with support for forward secrecy. In other words, the part of the connections that goes on the public Internet is protected by vanilla TLS. Once inside the enterprise or datacenter, the traffic is relayed to the final server over ETS, i.e., TLS with semi-static DH keys. Thus, the accusations against ETS seem to be partially ill-addressed. Note that while end users are unaware that their traffic is partially carried over ETS, they would also not know if it was done in plaintext (see Google's scenario described in NSA's MUSCULAR program [109]). This issue relates to the shift of the perceived "server-end" from an application server to an edge server.

Issues. ETS describes a key manager server that can either push or serve keys to enterprise servers and middleboxes. This aspect of the protocol aims to simplify the provisioning of keys. However, it significantly complicates deployments by adding an online key server that should properly authenticate the requests for keys. In turn, such new interfaces increase the attack surface. Also, ETS binds the TLS certificate to a hash of the static DH public key through a certificate extension. Thus, the validity period of certificates for ETS implicitly matches the use period of the DH key. In turn, those certificates should be rotated as well. In all three scenarios, ETS is or can be coupled with enterprise-trusted CA certificates, so the certificate rotation does not necessarily involve an external CA, which could otherwise pose as a bottleneck.

5.3.3 tls_visibility. An Internet draft [58] suggests a mechanism in TLS 1.3 for connections to be inspected by a pre-approved third party. Clients opt-in by advertising their willingness in the ClientHello. This proposal is suitable for inspecting TLS connections by enterprise middleboxes to a server within the enterprise or datacenter. The server is given a long-term ECDH public key DH_1^+ while authorized middleboxes are given the private counterpart DH_1^-. Upon an incoming TLS 1.3 connection with the empty tls_visibility extension present, the server generates a short-term ECDH key pair DH_2^+/DH_2^-, independently of the TLS key exchange (EC)DHE key pair. Then, the server calculates the shared secret Z as the result of the key exchange involving DH_1^+ and DH_2^- (requiring both keys to be selected from the same elliptic curve). The server responds with a ServerHello that includes the tls_visibility extension composed of DH_2^+ along with the early_secret and handshake_secret encrypted with a key derived from Z. Middleboxes are able to reconstruct Z using the provisioned DH_1^- and the given DH_2^+ from the ServerHello. Thereafter, the middleboxes can decrypt the TLS secrets, and derive the necessary keys to decrypt the rest of the traffic.

https://t.me/learningnets

Issues. It is easy for a malicious user to short-circuit an IPS that relies on the client's opt-in to receive the TLS secrets.

5.4 Client session key sharing

When splitting the TLS session is unacceptable at the client-side, some techniques and proposals similarly target TLS secret sharing at the client-side.

5.4.1 SSLKEYLOGFILE. Some browsers (e.g., Chrome, Firefox) can be asked to write TLS secrets to a log file [80] (e.g., PMS, MS, or handshake/application traffic secrets for TLS 1.3), which allow for passive decryption of the corresponding TLS traffic. This mechanism is generally enabled by setting the SSLKEYLOGFILE environment variable to a writable path. This approach has been adopted by some antivirus applications [65] by pointing the environment variable of browser processes to, e.g., a Windows named pipe.

Key exfiltration. By leveraging SSLKEYLOGFILE or by modifying the TLS implementation, keys can then be exfiltrated to a middlebox to provide filtering capabilities without splitting the TLS session, see LOCKS [13]. Commercial solutions for cloud environments also propose to scan a virtual machine memory on TLS handshakes to recover and aggregate session keys without application modification [85].

Issues. Support for this feature is limited to few applications and restricted (e.g., user warning, or special builds [80]) due to the obvious potential for misuse, i.e., this could enable easier and less privileged traffic-intercepting malware. Similar to DH key sharing on the server-side (§5.3.1), the key exfiltration requires a separate infrastructure. This is especially problematic on the client-side, arguably more hostile than a controlled server environment.

5.4.2 TLS-RaR. For IoT traffic monitoring, TLS-RaR [117] proposes to release TLS session keys for communications once they are finished. This schedule preserves the integrity of IoT traffic while making it auditable. This compromise may stimulate IoT vendors to open their opaque traffic.

Issues. Until the device releases past session keys, the end-user has no guarantee that all the current traffic will be auditable as there is no commitment or escrow mechanism.

6 DELEGATION OF CONTENT AND KEYS

An alternative set of approaches tailored for caching middleboxes (e.g., CDNs) delegates symmetric cryptographic operations to third-party-owned middleboxes in a way that they do not learn of the server's long-term key. This can be done with a varying degree of control by the server. By preventing the middlebox from distributing unapproved content, the server can retain full control. The server can also act as an online oracle for private key operations and give more freedom to the middlebox. Finally, the server can "vet" for middleboxes and remove itself from the communication unless necessary. Note that in the CDN context, the end-server is referred to as the origin server, while the CDN-owned node is called the edge node.

6.1 Server-controlled caching

We discuss below two main approaches for caching content with the server's collaboration and control.

6.1.1 Proxy encrypts, server authenticates. Lesniewski-Laas and Kaashoek [68] propose to "split" the encryption and authentication layers in TLS to allow untrusted collaborative proxies to distribute authenticated cached contents without the knowledge of the server's private key. The proxies are willingly participating, and do not need to be trusted to deliver the correct content. The protocol relies on the disclosure of the `server_write_key` by the origin server to the caching proxy, which enables the proxy to encrypt the cached content on behalf of the server. The missing authentication tags to form valid TLS records are provided by the server, which also encrypts

https://t.me/learningnets

the requested content as if it was fulfilling the request by itself; however, it only sends the corresponding tags to the proxy. From the client's perspective, there is no difference from a direct TLS session with the origin server.

The encryption layer is actually superfluous for the target goal. In fact, the authors point that an authentication-only TLS ciphersuite could be used; however, this would require the client to advertise such ciphersuite, which in practice is not the case for modern browsers, and is deprecated in TLS 1.3.

A patent by Akamai [48] proposes a similar idea aimed at selectively enabling a CDN to serve a request from its cache, without the knowledge of the server's private key. The main difference with Barnraising resides in the ability of the origin server to decide when to share the `client_write_key` to enable the edge server to decrypt client requests. If the edge server deems that it can serve the requested content from cache, it informs the origin server, which then shares the `server_write_key` and authentication tags as described above.

The patent also expresses the natural revocation of the write capability of the edge server since the origin can simply stop providing further MACs. The read capability allegedly can be revoked by requesting the client to renegotiate, effectively establishing new keys that the middlebox no longer knows. However, this mechanism seems insufficient to prevent a malicious middlebox from giving up read capability from the client, and read and write capabilities from/to the server (see details in Appendix A).

Issues. This scheme only works for non-AEAD ciphersuites for which the encryption and MAC keys are separate. This is not the case with PFS ciphersuites in TLS 1.2, and no longer possible in TLS 1.3. With AEAD, the edge server can also impersonate the origin server once it gets the encryption key since MACs are generated as part of the encryption. Also, the caching middleboxes only save network bandwidth from the origin, not computing resources.

6.1.2 Public/private content splitting. QoS3 [3] proposes to establish two connections to the server. One is end-to-end encrypted and delivers the main *private* document in which *public* content is tagged by the server (e.g., as an HTML tag attribute). Then, the public content can be fetched explicitly by browsers using a second proxied HTTPS connection that is intercepted by a caching middlebox. The integrity of the content is verified using pre-shared signatures from the main connection, or by leveraging the Subresource Integrity (SRI) mechanism [2]. This proposal improves on QoS2 [118] whereby the delegated connection is plain HTTP, a design justified by the fact that public resources may not need confidentiality guarantees. However, plaintext connections to the server are incompatible with HSTS policies.

Issues. The browser should withhold cookies from being sent over the proxied connection to avoid exposure of private content. Also, the solution only addresses caching for content served on the same domain.

6.2 Server-authenticated caching (content delegation)

6.2.1 Keyless SSL. The CloudFlare CDN proposes to serve TLS content without the knowledge of their customer's private key by requiring that the origin server also act as a key server that can sign and decrypt a key share or ePMS (for DHE and RSA key exchanges, respectively) upon request by the edge server. The edge server is then able to compute the shared secret and derive symmetric encryption keys with the client. The solution is dubbed as Keyless SSL [101].

Protocol flaws. The key server remains agnostic to TLS and simply acts as a signing/decryption oracle, communicating over a mutually authenticated channel with the edge server. In this setup, Keyless SSL was found to be vulnerable to two attacks when an attacker compromises an edge server [12]. First, for RSA handshakes, an attacker can query the origin key server from a compromised edge to decrypt previously captured ePMS, which leads to the full decryption of the corresponding traffic. Second, for DHE handshakes, the attacker can request a signature on a Server Config (SCFG) message as used in the QUIC protocol. The malicious SCFG message

https://t.me/learningnets

contains the attacker's DH parameters and a large expiration time. The attacker can then act as a QUIC server and impersonate the origin server to clients until the origin server certificate is revoked.

Bhargavan et al. [12] propose a solution to fix these issues by making the key server partially aware of TLS. In particular, it is the key server that generates and signs DHE parameters, and encrypts the Finished message given the transcript of the handshake between the client and the edge server. The edge does not know the master secret and is unable to resume the session by itself. The authors also discuss the need for an extra PKI to distinguish middleboxes and specific contents to be served. This solution requires three round trips with the origin server instead of one, thus comes at a significant performance cost.

Other issues. One main drawback of Keyless SSL is that the key server must be highly available, partially defeating the purpose of using a CDN, and adding one round trip between the edge and the origin servers per TLS handshake.

6.2.2 LURK. Limited Usage of Remote Key (LURK [75]) is a general protocol for context-dependent remote interactions with cryptographic material. Contexts are described in LURK extensions, e.g., for its use in TLS handshakes [74, 76]. LURK is positioned as an alternative to Keyless SSL, thus it is tailored for the CDN use case. The edge node requires an increased participation from the key server, e.g., for obtaining the server random value in DH key exchanges. Certain configurations of LURK appear not to be vulnerable to the attacks identified by Bhargavan et al. on Keyless SSL. In particular, the Freshness extension should limit the extent to which the key server can be used as an oracle.

Issues. LURK for TLS 1.3 is a lengthy Internet draft [76], which would require peer review and/or a formal treatment to be properly considered. In addition, due to the lack of known deployments and documentation, we do not further evaluate its features and benefits.

6.3 Server-vetted middlebox

Several approaches achieve content delegation by relying on special certificates with some degree of revocation, or on binding the server and CDN certificates.

6.3.1 Name constraints. The name constraint mechanism [29] restricts a CA to issue end entity certificates (EEC) for only a subset of names, e.g., subdomains only. These constraints can empower an enterprise to possess a CA certificate and issue EEC for its subdomains without the involvement of the CA. This mechanism could also be used by an end entity to issue new browser-trusted certificates for its domains only, which could be shared with CDNs. The proof of delegation becomes apparent from the chain of trust, and the end entity could revoke delegated certificates as needed. Importantly, CDNs do not learn of the EEC private key.

Issues. Due to poor business incentives from CAs, and insufficient support from browsers leading to wrongly accepted certificates,[2] name constraints CA certificates are uncommon. The end entity also needs to operate as a CA (e.g., revocation servers), although with limited responsibilities (its domains only).

6.3.2 Proxy certificates. Proxy certificates have been introduced in the context of Grid computing [30] to identify Grid users with a PKI. The purpose of such certificates was to enable the creation of dynamic identities (e.g., running jobs) and the delegation of some privileges to them (e.g., ability to access the user resources, or spawn new jobs).

RFC3820 [105, 116] standardizes Grid proxy certificates as X.509 certificates. This standard extends the X.509 PKI by allowing EEC to sign proxy certificates (PC) with their own key pair. PCs could be requested to the end entity by a second entity, making the end entity as a CA. PCs are also marked with a critical X.509 extension

[2]Name constraints are not properly enforced by all browsers [71], although the situation is improving, see https://nameconstraints.bettertls.com/

https://t.me/learningnets

called ProxyCertInfo. This extension holds a proxy policy, which specifies whether the PC inherits all or none of the privileges of the EEC, or whether an application-specific policy is present to define its privileges. A PC can issue another PC, subject to a separate PC path length constraint. In addition, certain restrictions apply: 1) the Digital Signature key usage bit is required on the issuing EEC; 2) the PC is chained to its issuing EEC (or another PC) by matching the PC issuer to the EEC (or PC) subject, and SANs are forbidden; 3) the PC subject name should be unique and start with the PC issuer name. RFC3820 proxy certificates are supported by OpenSSL since v0.9.8 [88].

Chuat et al. [23] propose a different semantic for proxy certificates, more adapted to delegation on the web. They borrow the idea behind the name constraint mechanism. Their proposed PCs work with regular non-CA EEC, alleviating the economic and poor support concerns on name constraints, and are constrained to the EEC SANs by default. The main use case for these PCs would be for CDNs to submit CSRs to the end entity for signature. CDNs could manage their own keys, and the origin server's private key is never shared.

Issues. The lack of SAN support in RFC3820 is prohibitive for modern web usages. Also, the revocation of certificate, and thus of the delegation intent, can only be done by pointing to the end entity as a CRL distribution point or OCSP server, which is impractical since the end entity is not a full-fledge CA. Chuat et al.'s proposal solve both problems. Notably, revocation is addressed by enforcing short-lived certificates that reduce the attack window after key compromise and enable a fine-grained control over the delegation period. However, it is not supported by clients out-of-the-box.

6.3.3 Delegated credentials. A relatively new Internet draft [10] proposes a variant of the proxy certificates defined in RFC3820, by only allowing an end entity with a special EEC to sign *delegated credentials* composed of a public key share and expiration time, bound to be used with TLS 1.3. The EEC is required to carry a special non-critical extension to permit the use of delegated credentials. When a client advertises support for this mechanism as part of the TLS extensions, the actual delegated credential is delivered as an extension in the CertificateEntry corresponding to the EEC in the Certificate message, and should be used in place of the EEC public key in the TLS connection (i.e., to verify the CertificateVerify message). This simplified approach avoids ambiguous interpretations of the semantics of X.509 proxy certificates.

6.3.4 Delegation through DANE. Liang et al. [71] propose to leverage DANE [37] to associate the server certificate with the CDN certificate used in the actual TLS connection. DANE is originally intended as a complimentary trust model for TLS that provides constraints on the website certificate in a TLSA DNS record. In this proposal, the server places its original certificate and that of the CDN itself in TLSA record, thereby asserting the delegation. When a client connects to the website through the CDN, it receives the CDN certificate valid for the CDN domain name only. The client then retrieves and validates the server certificate obtained through DANE and accepts the CDN certificate for the connection. DANE guarantees that the delegation is approved by the domain owner.

Issues. The support for DNSSEC is still not widespread and has been shown to be prone to misconfiguration [25].

7 HANDSHAKING WITH MIDDLEBOXES

When simple TLS splitting and key sharing solutions are not sustainable, in particular due to newly introduced security vulnerabilities [32, 38] and incompatibilities with TLS 1.3, another approach consists in including middleboxes in the TLS handshake, and assigning them different permissions or selectively disclosing parts of the traffic to them.

7.1 Selected disclosures

End-to-end Fine Grained HTTP (EFGH [45]) enables policy-based disclosure of encrypted content to a middlebox, and authenticated changes. During the TLS handshake, four symmetric keys are derived: one encryption key G

https://t.me/learningnets

known to all parties, another one K_{CS} known to the endpoints only, and two endpoint-to-middlebox keys K_{CP} and K_{PS} for authentication. The application records are encrypted with G when it is desirable that the content be visible to the proxy, otherwise the end-to-end key K_{CS} is used. K_{CS} also serves to authenticate the records in both cases. When the proxy modifies the traffic, e.g., to deliver content from its cache on behalf of the server, it encrypts the records with G and authenticates them using K_{CP} or K_{PS} depending on the endpoint.

Issues. The protocol requires changes in the TLS record format to accommodate extra headers and metadata, indicating e.g., whether the record is intended to be decrypted by the proxy. In turn, this leads to changes in the client and server implementations. Also, EFGH can only accommodate one middlebox.

7.2 Permission contexts

Multi-Context TLS (mcTLS [83]) proposes to establish various contexts protected by different sets of symmetric encryption and MAC keys. Each context is managed at the application layer, and corresponds to different middlebox access privileges. Each endpoint performs a key exchange with each middlebox, optionally authenticated. The middleboxes are given half of the context keys by each endpoint for each context they are included in, thus requiring the agreement of both the client and server to access a context. There are two levels of privileges: reader, which receives the reader encryption and MAC keys (for verification only); and writer, which is also given the writer MAC keys (for content modification).

The protocol adds the MiddleboxListExtension to the ClientHello, which lists the middleboxes to be included, the labeled contexts and their middlebox permissions. Following the response from the server, each middlebox also responds to each endpoint with their certificate and endpoint-specific key shares (in the case of DHE). In turn, the client is able to derive a distinct shared secret for each middlebox and the endpoint.

The endpoints then derive the endpoint keys that are used to authenticate the traffic and verify that no modification was performed. They also derive half the reader and writer keys, and send them in MiddleboxKeyMaterial messages to the appropriate middleboxes, encrypted with their corresponding shared secrets. The middleboxes reconstruct the context keys. An optional feature allows the server to let the client send the whole context keys to the middleboxes. Writers are able to detect when readers illegally modify the content, while readers can only tell if modifications came from other external entities (i.e., an attacker). The actual context traffic is encrypted using the reader encryption key; however, endpoints also add MACs to TLS records calculated with the three MAC keys.

The contexts can then be used at the application/server discretion. For instance, HTTP requests and responses could be split into different contexts to enable compression middleboxes to act on the server's response only. Similarly, headers and bodies could be split to enable tracker blockers to perform URL-based filtering without access to the request body and page content. Finally, browsers could decide to enable compression proxies for media content when using metered or slow connections, then dynamically revert to a no-middlebox context when conditions are more favorable.

The authors leave out the provisioning and authentication of authorized middleboxes, and subsequent user interface issues, but point to possible deployment options.

Protocol flaws. Bhargavan et al. [11] describe two flaws in mcTLS, originating from the lack of formal verification of the protocol. Both attacks assume an attacker who can intercept the traffic on both sides of a middlebox.

The first flaw exploits the absence of a full handshake between the middleboxes and the endpoints, i.e., there are no Finish messages for middleboxes to verify that messages have not been tampered. This allows an attacker who can tamper and revert the ClientHello and ServerHello as they pass through a middlebox to confuse the said middlebox. One attack consists in changing the server name in the SNI extension and returning the corresponding certificate, to lure the middlebox into applying different filtering rules. If the middlebox is additionally caching content, it will misattribute the server response to the wrong server, leading to cache poisoning.

https://t.me/learningnets

The second flaw exploits the ability of reader middleboxes to modify the content and calculate a valid MAC (as they are given the reader MAC key), which can confuse other readers down the path before being rejected by the endpoints.

Those flaws consider a powerful attacker with the ability to intercept network communications simultaneously at different locations between the endpoints, which may not be realistic in certain cases.

TLMSP. Transport layer Middlebox Security Protocol [40] is an ETSI draft based on mcTLS. It aims at standardizing the idea of multiple contexts with the ability to let middleboxes read and insert or delete information. Furthermore, it brings on-path middlebox discovery where mcTLS suggested to pre-configure the clients. Also, it aims to fix the previously reported flaws by introducing a new Finished message (MiddleboxFin) to conclude the handshake between the endpoints and each middlebox. Finally, other improvements include a guarantee that the messages went through all middleboxes and in the intended order. This property is also guaranteed in maTLS (see §7.3). Note that the standard is not yet finalized, thus we do not further evaluate this protocol.

7.3 Publicly auditable middleboxes

Unlike mcTLS, Middlebox-Aware TLS (maTLS [66]) proposes to negotiate separate TLS sessions between middleboxes instead of sharing the same session secrets. However, contrary to simply splitting the sessions (§5.1), maTLS enables both endpoints to supervise each segment's negotiated parameters, and for the client to authenticate both the middeboxes and the end-server. This architecture also enables the endpoints to detect which specific middlebox modified the traffic, even when two of them have write access.

During the ClientHello/ServerHello exchange, the participating middleboxes append their certificate to the server Certificate message, and their DH key share. The client is able to validate each certificate. Each endpoint and middlebox establishes a TLS session with the next entity. Then, both endpoints generate a shared secret with each middlebox (and the other endpoint, as in TLS). The middleboxes derive pairwise accountability keys with the endpoints, which are used to prove the security parameters negotiated between each segment. When forwarding or modifying a message, a middlebox appends a Modification Log (ML) to the message, which consists of a chain of HMACs on the message received and the one sent by each middlebox. The client is able to verify which middlebox modified the message, and that the message went through all the middleboxes in the right order.

Middlebox service providers (MSP) should obtain a valid publicly trusted certificate with critical extensions to communicate to the endpoints whether the middlebox requires read or write access to the traffic. Middlebox certificates are logged into Middlebox Transparency (MT) logs, similar to CT logs, to provide auditability of issuance.

Issues. The authors claim that maTLS "allows middleboxes to participate in the TLS session in a visible and auditable fashion." However, maTLS does not achieve auditability of *participation in a TLS session*, which could be interpreted as the ability to detect illegitimate traffic inspection after-the-fact. Rather, an audit of MT logs only reveals which MSP has been issued a certificate (which could be viewed as a "license to intercept"). Consequently, while each middlebox is implicitly vetted by the CA that issued its special-purpose certificate, the end user is still involved in the decision process to accept those certificates. Given that middlebox certificates cannot be bound to a unique and unambiguous identity, the end-user is vulnerable to phishing-like attacks whereby an attacker obtains a certificate with a convincing name, or even the same name as the expected MSP's by justifying the name in another jurisdiction (cf. [51]). Paradoxically, by enforcing reliable identification of the providers, small enterprise middleboxes and every instance of client-end software solutions (e.g., antivirus) would be prevented from obtaining such certificates, making maTLS either as a non-usable or incomplete protocol.

https://t.me/learningnets

8 PRIVACY-PRESERVING INSPECTION

A different approach to traffic inspection consists in limiting the exposure of data to middleboxes by providing evidence to the endpoints that the operations performed are trustworthy instead of blindly trusting middleboxes once they gain read and or write capabilities.

8.1 Searchable encryption-based solutions

Techniques based on searchable encryption have been proposed to preserve the privacy of filtered traffic.

BlindBox [100] enables for middleboxes to perform basic keyword searching and pattern matching on TLS traffic without revealing all of the data to the middleboxes. The authors consider two privacy models that enable different types of operations. In the exact match privacy model, only keyword matching is possible while not revealing the rest of the data. In the probable cause privacy model, the traffic can be decrypted for further processing (e.g., regular expression matching) if a keyword is found. The filtering rules are provided by a trusted third party, and agreed upon by each sender during the connection establishment that leverages garbled circuits and oblivious transfer. BlindBox requires each sender to tokenize, encrypt and send their traffic separately from TLS, for the middlebox to analyze.

Issues. At least one receiver should be honest and detect whether the TLS and BlindBox traffic differ; otherwise, the middlebox can be lured into inspecting fake data. Also, this solution incurs a 500% bandwidth overhead, and requires 97 seconds to perform the initial TLS handshake for a typical IDS solution with 3,000 rules.

The use cases presented by the authors include the detection of malware installed on student laptops connected to the university network, and the filtering of traffic by an ISP/enterprise in accordance to the user's preference (e.g., parental control, document watermarking). We note that in the former case, the malware needs to comply with the BlindBox protocol, which is unlikely. Indeed, both the malware and the remote server may be controlled by the attacker, defeating BlindBox's threat model. In the latter case, BlindBox needs to be widely deployed on web servers for the user to truly benefit from the filters.

Embark [64] extends BlindBox for outsourced middleboxes typically located in the cloud, which additionally must not learn about the origin and destination IP addresses; thus they can not route the traffic. The performance of BlindBox is drastically improved in PrivDPI [84] with faster rule encryption algorithm.

8.2 Using Trusted Execution Environments

Rather than involving cryptographic solutions, other proposals rely on the guarantees of Trusted Execution Environments (TEEs) such as SGX with Remote Attestation, to protect long-term keys or even the whole middlebox operations while operating on the traffic itself with low overhead.

SGX-Box [54] proposes to run middlebox capabilities in SGX enclaves. It relies on memory isolation to keep the computations private, and on remote attestation to guarantee that the middlebox program executed is the intended one. Moreover, to abstract packet parsing and TLS decryption operations, SGX-Box introduces a high-level language to perform middlebox operations.

The authors propose that the remote attestation is performed by the end server, which then shares TLS session secrets out-of-band with the SGX-Box middlebox. This design requires the server's involvement and approval of the middlebox. Clients could also request an attestation; however, it is unclear how end-users or enterprises enforce the use of specific middleboxes when the server is responsible for sharing TLS secrets.

Similar constructions that rely on Intel SGX have been proposed. STYX [115] provisions the private key in an SGX enclave, mbTLS [82] enables endpoints to communicate per-segment keys to middleboxes after the software running in enclaves is remotely attested. More recently, Phoenix [55] develops *conclaves*, containers of enclaves, to port legacy applications in enclaves. However, SGX-provided guarantees have been eroded by numerous attacks, see [17, 20, 108].

https://t.me/learningnets

Table 3. Stakeholder alignment and conceivable applicability of approaches to enable content access according to middlebox use cases. Mismatching incentives lead to a gray circle. We consider the end-user organization imposes its incentives on end-users and clients in use cases *d*, *f*, *h*-CN and *j*-CN. Legend: Circle portioning qualifies the middlebox use case applicability. ●=applicable, ◐=partially applicable, ◑=potentially applicable but is not proposed as such (proposals only), ○=not applicable. CN=Client network, SN=Content provider (server) network.

	Discussed in §	Scheme name \ Use cases	*a,b,c.* ISP legal operations	*d.* Fraud detection	*e.* Malware download prevention	*f1.* IDS on outgoing connection	*f2.* IPS on outgoing connection	*g1.* IDS on incoming connection	*g2.* IPS/WAF on incoming conn.	*h.* Data retention (compliance)	*i.* Pre-request authorization	*j.* Data loss prevention	*k1.* IoT dev. monitoring (server auth.)	*k2.* IoT dev. monitoring (mutual auth.)	*l.* L7 load-balancing	*m.* Broadcasting	*n.* Server-mandated caching	*o.* Opportunistic caching	*p.* Transcoding/compression	*q1.* Problem troubleshooting (user help)	*q2.* Problem troubleshooting (no user help)	*r.* Parental control	*s.* ISP billing/tracking
		Middlebox location →	ISP	CN	CD	CN	CN	CN	SN	SN	CDN	CN	SN	CDN	CN	SN	CN	ISP	CDN	SN	ISP	CDN	CN	ISP	ISP	CDN	SN	CD	CN	ISP	ISP
Session splitting/key sharing		TLS session splitting	●	●	●	●	◐	●	◐	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●	●	●	●	●	●	●	●
	5.1	w/ ProxyInfoExtension [72]	●	●	●	●	◐	●	◐	●	●	●	●	○	●	●	●	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●
		w/ client-trusted CA	●	●	●	●	◐	●	◑	●	●	●	●	○	●	●	●	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●
	5.2	w/ server certificate	●	●	●	●	◖	●	◑	●	●	●	●	○	●	●	●	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●
	5.2	Certificate sharing (passive)	●	●	●	●	●	●	●	●	●	●	●	○	●	●	●	○	○	○	○	○	○	○	○	○	○	●	●	●	●	●
	5.3.1	Static DH key sharing (server)	●	●	●	●	●	●	●	●	●	●	●	○	●	●	●	○	○	○	○	○	○	○	○	○	○	●	●	●	●	●
	5.3.2	ETS [41] (scenario 1&3)	◑	●	○	○	●	○	○	○	○	●	○	○	●	○	◑	○	○	○	○	○	○	○	○	○	○	○	○	○	◑	◑
		ETS [41] (scenario 2)	○	○	○	○	○	○	●	○	○	○	●	○	○	●	○	○	○	○	○	○	○	○	○	○	○	●	●	○	◑	○	○
	5.3.3	tls_visibility [58]	◑	○	○	○	○	○	◑	◑	◑	○	○	○	○	◑	◑	○	○	◑	○	○	○	○	○	○	○	●	◑	○	○	○	○
	5.4.1	SSLKEYLOGFILE [80]	○	◐	●	○	○	○	○	○	○	○	○	○	◐	○	◑	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
	5.4.1	LOCKS [13]	◑	●	●	●	●	●	◑	◑	◑	●	◑	○	●	◑	◑	○	○	◑	◑	◑	◑	○	○	○	○	●	●	●	●	●	◑
	5.4.2	TLS-RaR [117]	◑	◑	○	○	◐	○	◖	○	○	◑	◑	○	○	◑	◑	○	○	○	○	○	○	○	○	○	○	◑	◑	○	○	○	○
Delegation	6.1.1	Auth/encryption splitting [68]	◑	○	◑	◑	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	◑	◑	◑	◑	○
	6.1.1	Akamai's patent [48]	◑	○	○	○	○	○	◑	◑	◑	○	◑	○	○	◑	◑	○	○	○	●	○	○	○	○	◑	◑	◑	○	○	○	○	○
	6.1.2	QoS2 [118], QoS3 [3]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	6.2.1	Keyless SSL [101]	◑	◑	◑	◑	◖	◑	◑	◑	◑	◑	◑	○	◑	◑	◑	○	●	◑	◑	◑	◑	◑	◑	◑	◑	◑	◑	◑	◑	◑	◑
	6.3.2	Proxy certificates [23]	●	●	●	●	◖	●	◖	●	●	●	●	○	●	●	●	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●
	6.3.3	Delegated credentials [10]	●	●	●	●	◖	●	◖	●	●	●	●	○	●	●	●	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●
	6.3.4	Delegation over DANE [71]	●	●	●	●	◖	●	◖	●	●	●	●	○	●	●	●	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●
3-way	7.1	EFGH [45]	●	●	●	●	◖	●	◖	●	●	○	●	●	●	●	●	○	●	◑	◑	●	●	●	●	●	●	●	●	●	●	●
	7.2	mcTLS [83]	●	●	●	●	◖	●	◖	●	●	○	●	●	●	●	●	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●
	7.3	maTLS [66]	●	●	●	●	◐	●	◖	●	●	◑	●	●	●	●	●	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●
	8.1	BlindBox [100], PrivDPI [84]	○	◑	●	●	●	●	◑	◑	◑	○	◑	○	◑	◑	○	○	○	○	○	○	○	○	○	○	○	○	◑	◑	◑	◑	○
	8.2	SGX-Box, mbTLS [54, 82]	◑	○	○	○	○	○	●	●	●	○	◑	○	○	○	○	○	◑	○	○	○	○	○	○	○	◑	◑	◑	○	○	○	○
	8.2	STYX, Phoenix [55, 115]	◑	○	○	○	○	○	●	●	●	○	◑	●	○	◑	○	○	◑	○	○	○	○	○	○	○	◑	◑	◑	○	○	○	○

9 EVALUATION

In this section, we evaluate the consistency of the techniques and proposals discussed in §5,6,7,8 against the actual use cases we identified in §4 for access to HTTP traffic in the presence of TLS. We consider the compatibility of these schemes with participating stakeholders' incentives evaluated in 4.3. Second, we set out an evaluation framework aimed at characterizing the deployability of those schemes along with their security features and guarantees, and run each scheme through it.

9.1 Disincentives for use cases

We evaluate in Table 3 how techniques and proposals match with the stakeholders' incentives for each use case of a middlebox. This evaluation aims to identify proposals that are ill-suited for specific use cases due to obvious

https://t.me/learningnets

misalignment of incentives. In particular, it does not account for technical deployability concerns and security features, which are further evaluated in §9.2. The criteria for applicability only considers whether the scheme enables the required access to unencrypted content for the use case. For instance, it is conceivable that proxy certificates are applicable for the *IDS on outgoing connections (f1)* use case, despite the content provider's obvious reluctance to provide such certificates to the EUO in the general case.

For general techniques such as session splitting and certificate sharing, we consider the applicability against all use cases, despite foolish combinations, e.g., session splitting at the server-side that requires provisioning the client with a custom root certificate (a situation typically seen at the client-side). This serves as a comparison for other proposals.

From this simple evaluation, we are able to make the following observations.

Lack of incentives invalidates several proposals. From Table 3, it is apparent that many seemingly applicable scheme-use case pairs become ill-suited due to a lack of incentives from one of the main stakeholders. Irrespective of further security and deployment evaluations, this analysis shows that some proposals, although promising on other aspects, are likely stopped from even being considered due to their design. In particular, schemes that require the collaboration of both endpoints for all applicable use cases are deployment-unfriendly. Such schemes could only be beneficial if the sum of the use cases they solve may convince all involved stakeholders to support them. For instance, BlindBox is targeted at filtering traffic using third-party rules selected by the client; however, it requires the server's active participation. Conversely, tls_visibility cannot achieve any use case beyond problem troubleshooting with the user's help since it mandates the client's participation for server-side middlebox operations. The same applies to delegation through special certificates (e.g., proxy certificates, delegated credentials); however, see further discussion in §10.1.

No universal solutions. EFGH, mcTLS and maTLS are presented as universal solutions to accommodate any sort of common middlebox [45, 66, 83]. However, EFGH and mcTLS require the support of the server and all three require support of the client as well, which is incompatible with the incentives of the end-user, their organization, or the content provider. For instance, to support malware download prevention, mcTLS requires the support from the server, which is mostly uninterested in this task. As a consequence from the previous point, these general-purpose solutions do not achieve the intended goal of giving more control over the presence of middleboxes; rather, they force an unrelated endpoint to support its solution with no clear benefit. They would only be useful if they replaced TLS. Arguably, such a move deserves strong justifications. Of interest, maTLS benefits from a per-segment construction (a "bottom-up" approach as the authors describe), which enables use cases where middleboxes are operated near the client-side. Although the client does not fully benefit from the protocol's features when the server does not support it, middleboxes can be accommodated.

No convincing scheme for ISP legal operations. All schemes that could in theory apply to ISP legal use cases are obviously incompatible with the end-user's incentives as the client needs to support the scheme (e.g., session splitting with non-browser trusted certificate), and with the server operator's incentives if required to participate (e.g., mcTLS). Although TLS session splitting w/ client-trusted CA achieves the technical need, it is incompatible with CA certificate issuance policies.

Unaddressed use cases. Consumer IoT device monitoring suffers from a lack of incentive from the main stakeholder, i.e., the vendor, and therefore no scheme seems fit for this purpose. It seems necessary that a business incentive (e.g., through certifications) or legal requirements be created for IoT device vendors to open interfaces that would let end-users monitor the traffic originating from these devices. Also, broadcasting and L7 load balancing are special use cases for which little to no scheme applies.

https://t.me/learningnets

9.2 Evaluation framework

We describe here the criteria that enable the evaluation of the proposals and techniques discussed in the previous sections. As various objectives require different criteria, we focus our evaluation on three main categories of use cases: CDN-type (N) middleboxes (*g*, *i*, *l*, *n* and *p*), client-side (C) and server-side (R) middleboxes. Our evaluation is based on general and case-specific criteria for deployability, security, usability and extra features provided.

9.2.1 Deployability. Schemes that can be readily deployed while being compatible with existing up-to-date major browsers fulfill *Client Compatibility (D1)*, while those that require minor modifications to the clients (typically supporting an additional TLS extension), partially fulfill it. Conversely, *Server Compatibility (D2)* is fulfilled when schemes are compatible with existing servers. We do not expect schemes to fulfill both criteria, and decide which one to evaluate depending on the use case. Also, schemes that do not require any changes in the application running on top of TLS (client or server-side) fulfill *No Application-layer Rewrite (D3)*. If changes are only needed to better take advantage of the scheme, they partially fulfill the criteria. When schemes enable the client with technical means of knowing about the presence of middleboxes owned or operated by a third-party fulfill *Client Awareness (D4)*. A full mark is given if the client can fully distinguish all middleboxes; half mark if the client may only know if any number of middleboxes is present or not. Some schemes may rely on features that are not widely supported such as DNSSEC on the client, while others fulfill *No Unsupported Prerequisite (D5)*. Requiring SGX on the middlebox partially fulfills the criteria. Schemes where the establishment of a secure connection by the client does not require additional connections to exchange side information with middleboxes or the other endpoint fulfill *No Add. Connections (D6)*. When an endpoint and middlebox maintain a permanent connection used for all TLS connections, they partially fulfill D6. Schemes that preserve the existing PKI and do not involve CAs and CT logs in extra activities fulfill *No Extra Burden on PKI (D7)*. If schemes only require new or unusual X.509 extensions or parameters in TLS certificates issued by trusted CAs, they partially fulfill the criteria. When the scheme does not introduce extra servers or new roles for existing servers, it fulfills *No Suppl. Infrastructure (D8)*. Schemes that only introduce a reasonable network bandwidth and computation overhead fulfill *Reasonable Overhead (D9)*. Here, reasonable means that the overhead does not appear to be prohibitive for deployment and user experience.

9.2.2 Security and privacy. Simple schemes leverage the knowledge of the server's certificate private key or other long-term keys. When those secrets are not exposed to third-party-owned middleboxes, schemes fulfill *Long-term Secret Stays Private (S1)*. Ciphersuites recommended in TLS 1.2 and all those in TLS 1.3 comprise AEAD symmetric ciphers. When schemes can encrypt the request and content served by the end-server and middleboxes with these ciphers, they fulfill *Support for AEAD (S2)*. Similarly, recommended key exchanges in TLS 1.2 and all those in TLS 1.3 are based on Diffie-Hellman. When schemes truly leverage ephemeral keys in all key exchanges from which derived keys are used to encrypt the request and content, they fulfill *Support for FS (S3)*. Schemes that allow the client to verify and eventually choose the ciphersuite (CS) selections on all segments between both ends fulfill *Client CS Supervision (S4)*. Schemes that enable the client to see and validate the end-server certificate fulfill *End-server Authentication (S5)*. The connection should be cryptographically bound to this certificate to receive a full mark, i.e., encryption secrets must be derived from a key exchange authenticated (directly or not) by that certificate's private key. Schemes where the middlebox is trusted to expose the end-server certificate receive a half mark. If there is a way for the most interested end party to verify or limit operations done by a middlebox on the traffic, to ensure that, e.g., it does not retain or process data in an unexpected way, the schemes fulfill *Operation Verifiability (S6)*. This can be provided by cryptographic means, or through trusted execution environments.

Sub-case: CDN. In the event of an edge server compromise, schemes that prevent the attacker to break past recorded communications made through this or other edges fulfill *FS Despite Edge Compromise (N-SA)*. Schemes

https://t.me/learningnets

that enable the client to validate the end-server certificate (S5), and verify that the CDN delegation is approved by the content provider fulfill *Origin & CDN Authentication (N-SB)*. The connection should be cryptographically bound to both verifications. When the origin server can control and restrict all content served by the CDN, schemes fufill *Origin-controlled Delivery (N-SC)*. Note that N-SC is a subset of S6.

Sub-case: Client/server-side solutions. The content provider is the only one to decide of its own architecture, including middleboxes. When the client cannot opt out of those middleboxes, schemes fulfill *No Bypass of Server-side MB (R-SA)*. When the traffic is modified by a client-side middlebox, schemes fulfill *Modification Accountability (C-SA)* if the modification is easily detectable by mechanisms provided in the protocol (e.g., verification of a cryptographic hash), and the middlebox responsible for the modification is unambiguously identifiable (provided all middleboxes are first authenticated). Also, a middlebox that is only allowed to read the traffic cannot modify it without being detected in a similar fashion. If the identification of the middleboxes only return the set of middleboxes that are allowed to modify the traffic (e.g., those with the ability to modify the traffic), the proposal receives a half-mark.

9.2.3 Usability and other options. Ideally, the end-user should not need to make new security-related decisions, due to the inherent risk that users make the wrong choice. Schemes that do not involve the user into making new decisions related to middleboxes fulfill *No User Involvement (U1)*. When schemes can help to solve other strong problems in the TLS ecosystem, they fulfill *Extra Benefits (O1)*. For instance, proxy certificates and delegated credentials enable short-lived certificates/credentials to be issued without the involvement of CAs, thereby alleviating the need for (problematic) revocation mechanisms. Schemes that are compatible with TLS client authentication fulfill *Support for TLS Client Auth (O2)*.

Sub-case: Client/server-side solutions. Permanent client-side middleboxes can be configured on end-user systems to be trusted, in which case schemes fulfill *Pre-configured Trust (C-OA)*. When unknown middleboxes can be discovered by the schemes, they fulfill *Middlebox Discovery (C-OB)*. If a scheme can accommodate multiple middleboxes in the connection, it fulfills *Multi-MB Support (C-OC)*. The scheme fulfills *Private Mode (C-OD)* if the client can opt out of inspection for selected connections.

10 DISCUSSION AND CONCLUDING REMARKS

We list below important findings and discuss open questions that arise from our analysis.

10.1 Findings to highlight

Generic solutions do not solve any use case well. Table 3 shows the misalignment of stakeholders incentives that affects in particular generic solutions such as EFGH, mcTLS, and maTLS. These generic solutions require support from both ends and are therefore less likely to be deployed. Moreover, they lack essential properties in specific use cases. For instance, server-side middleboxes are completely bypassable if the client decides not to support the protocol (R-SA in Table 4). Unless these new proposals are universally deployed, users could easily opt to use regular TLS instead. A better option seems to develop solutions tailored for specific use cases and that are aligned with stakeholder goals.

Lack of well-defined goals. From our reading of the proposed protocols, different research proposals seem to take many directions, sometimes without precise objectives, and often address different problems, which complicates comparative evaluation. Without clearly specified goals, it is difficult to evaluate whether a proposed scheme meets the objectives. Our discussion of practical use cases for access to plaintext (§4) is intended to address this issue.

Two main approaches for CDNs. The CDN use case is approached in two different ways in the literature: either give more autonomy to the middlebox without releasing the origin's long-term secret, or control the

https://t.me/learningnets

Table 4. Comparative evaluation of schemes considering three use cases: N for CDN, R for server-side, C for client-side security middleboxes, * means all three. Notation for entries: ●=property fulfilled by a scheme fulfilled; ◐=partially fulfilled; ○=not fulfilled. Shaded regions mean property not applicable to specified use case.

Discussed in §	Scheme name \ Criteria	Use Case	D1. Client Compatibility	D2. Server Compatibility	D3. No Application-layer Rewrite	D4. Client Awareness	D5. No Unsupported Prerequisite	D6. No Add. Connections	D7. No Extra Burden on PKI	D8. No Suppl. Infrastructure	D9. Reasonable Overhead	S1. Long-term Secret Stays Private	S2. Support for AEAD	S3. Support for FS	S4. Client CS Supervision	S5. End-server Authentication	S6. Operation Verifiability	N-SA. FS Despite Edge Compromise	N-SB. Origin & CDN Authentication	N-SC. Origin-controlled Delivery	C-SA. Modification Accountability	R-SA. No Bypass of Server-side MB	U1. No Risky Decisions	O1: Extra Benefits	O2: Support for TLS Client Auth	C-OA. Pre-configured Trust	C-OB. Middlebox Discovery	C-OC. Multi-MB Support	C-OD. Private Mode
	TLS session splitting	*	○	●	●	●	●	●	●	●	●	●	●	○	○	○	◐	○	○	○	●	●	○	○	●	○	◐	◐
5.1	w/ ProxyInfoExtension [72]	*	○	●	●	●	●	●	●	●	●	●	●	○	◐	○	◐	◐	○	○	●	●	○	○	●	○	◐	◐
	w/ client-trusted CA	*	●	●	●	○	●	●	◐	●	●	○	●	●	○	○	○	◐	○	○	○	●	●	○	○	○	○	●	◐
5.2	w/ server certificate	*	●	●	●	○	●	●	●	●	●	○	●	●	○	◐	○	◐	○	○	○	●	●	○	○	○	○	●	◐
5.2	Certificate sharing (passive)	R	●	●	●	○	●	●	●	●	●	○	●	○	●	●	○					●	●	○	●				
5.3.1	Static DH key sharing (server)	R	●	◐	●	◐	●	●	●	●	●	○	●	○	●	●	○					●	●	○	●				
5.3.2	ETS [41] (scenario 1)	R	◐	○	●	◐	●	●	◐	○	●	◐	●	○	●	●	○					●	●	○	●				
	ETS [41] (scenario 2)	R	●	○	●	○	●	●	◐	○	●	○	●	○	●	●	○					●	●	○	●				
	ETS [41] (scenario 3)	R	○	●	●	●	●	●	●	○	●	●	●	○	○	●	○					●	●	○	●				
5.3.3	tls_visibility [58]	N,R	○		●	●	●	●	●	●	●	●	●	●	●	●	○	●	○	○		○	●	○	●				
5.4.1	SSLKEYLOGFILE [80]	C	○	●	●	●	●	●	●	●	●	●	●	●	●	●	○						●	○	●	●	○	◐	○
5.4.1	LOCKS [13]	C	○	●	●	●	●	◐	●	○	●	●	●	●	●	●	○						●	○	●	●	○	◐	○
5.4.2	TLS-RaR [117]	C	○	●	●	●	●	●	●	●	●	●	●	●	●	●	○						●	○	●	●	○	◐	○
6.1.1	Auth/encryption splitting [68]	N	●		●	○	●	●	●	○	●	●	○	●	●	●	○	●	○	●			●	○	●				
6.1	Akamai's patent [48]	N	●		●	○	●	●	●	●	●	●	○	○	●	●	○	●	○	●			●	○	●				
6.1.2	QoS2 [118], QoS3 [3]	N	○		○	●	●	○	◐	●	●	●	●	●	○	○	○	●	○	●			●	○	○				
6.2.1	Keyless SSL [101]	*	●		●	○	●	○	●	○	●	●	●	●	○	●	○	○	○	○			●	○	○				
6.3.1	Named constraints [29]	*	◐	●	●	●	◐	●	◐	●	●	●	●	●	○	●	○	◐	●	○	○	●	●	●	●	○	○	●	◐
6.3.2	Proxy certificates [23]	*	○	●	●	●	●	●	●	●	●	●	●	●	○	●	○	◐	●	○	○	●	●	●	●	○	○	●	◐
6.3.3	Delegated credentials [10]	*	◐	●	●	●	●	●	◐	●	●	●	●	●	○	●	○	●	●	○	○	●	●	●	●	○	○	◐	●
6.3.4	Delegation over DANE [71]	*	○	●	●	●	○	●	○	●	●	●	●	●	○	●	○	◐	●	○	○	●	●	○	●	○	○	◐	◐
7.1	EFGH [45]	*	○	○	◐	●	●	●	●	●	●	●	●	●	●	●	○	●	●	○	●	○	○	○	●	●	○	○	●
7.2	mcTLS [83]	*	○	○	◐	●	●	●	●	●	◐	●	●	●	●	●	○	◐	●	◐	◐	○	○	○	●	●	○	●	●
7.3	maTLS [66]	*	○	◐	●	●	●	●	○	●	●	●	●	●	●	●	○	◐	●	◐	●	○	○	○	●	●	●	●	●
8.2	SGX-Box [54]	N	●		●	○	◐	○	●	○	●	●	●	●	●	●	●	●	○	●			●	○	●				
8.2	STYX [115]	N	●		●	○	◐	◐	●	○	●	◐	●	●	●	●	○	◐	○	○			●	○	●				
8.2	mbTLS [82] (server-side MB)	N,R	●		●	○	◐	●	●	●	●	●	●	●	●	●	●	●	○	●		●	●	○	●				
8.2	mbTLS [82] (client-side MB)	C	○	●	●	○	◐	●	●	●	●	●	●	●	●	●	●				○		●	○	●	●	●	●	●
8.2	Phoenix [55]	N	●		●	○	◐	◐	●	●	●	◐	●	●	●	●	●	◐	○	●			●	○	●				

operations of the middlebox through a Trusted Execution Environment (TEE) to only release origin secrets to a secure enclave. These two approaches highlight an underlying difference in trust assumptions. Which of these two approaches is better would appear to depend on specific context.

Performance-enhancing proxies cannot be reliably discovered. The discovery or automated provisioning of client-side middleboxes (up to the ISP included), e.g., for network-dependent optimization purposes, is a challenging task that is often overlooked or considered orthogonal by researchers. For example, while mcTLS

https://t.me/learningnets

authors suggest to provision proxy parameters through DHCP (arguably an insecure design), maTLS relies on CAs to issue special and unrestricted publicly trusted middlebox certificates; neither of these approaches seem securely deployable. EFGH is silent regarding provisioning. This leaves us with incomplete solutions.

Delegated credentials offer to solve multiple problems. In Nov. 2019, the delegated credentials (DC) mechanism was implemented by CloudFlare and Firefox (in nightly builds), with the support from DigiCert [102]. The early adoption of DC, despite a misalignment of stakeholder incentives in our view (Table 3), is due to another advantage of this mechanism. In practice, DC brings the equivalent of short-lived certificates, at low cost, which mitigates the certificate revocation issue by reducing the attack window from months or years to days [23]. Proxy certificates bring a similar benefit. Thus, this may indicate that proposals addressing other open issues may succeed by solving multiple hard problems at the same time. Interestingly, we note that DC is conceptually simpler compared to academic efforts.

TLS 1.3 analysis efforts are not yet matched. Formal verification of protocols led to the discovery of vulnerabilities in Keyless SSL [12] and mcTLS [11]. It is unclear whether other proposals have received such peer scrutiny. Given the complexity of TLS and the formal treatments it received, we expect that any significant modifications to the protocol may introduce weaknesses if not carefully reviewed by the community and verified by e.g., formal methods. With the numerous proposals in the literature, we hope our study will serve to target effort towards the most promising ones in priority. On this path, it is worth highlighting that maTLS authors leverage a verification framework tailored for security protocols [73].

Usability concerns. For client-side middleboxes that require the client's approval or validation, the end-user interface and experience is poorly studied. For instance, should the URL bar display separate secure locks in the browser for each middlebox? Should the user consent to the middlebox for each connection, or should she be given the choice to enable/disable the middlebox? If a middlebox certificate is expired, can the user bypass the browser warning, and for how long? Such issues further complicate HTTPS usability, and are under-studied.

10.2 Stronger threat models vs. simpler designs

Concerns about man-in-the-middle attacks on TLS and middlebox-friendly proposals loosely assume that the attacker is well-located to actively monitor and tamper with communications. However, certain attacks are only possible when the attacker is present at multiple points on the communication path, often in very different geographic locations. For instance, flaws considered [11] in mcTLS and subsequently addressed in maTLS [66] require the attacker to manipulate traffic just before and right after it goes through a middlebox. This may effectively translate into two network attacks: one near the client, and another one near the server. This threat is significantly more advanced than with a simple coffee shop-level attacker, and the additional protections against such threats bring an extra cost that could be overwhelming in view of actual risks. However, since TLS is not vulnerable to such advanced network adversaries, it seems unreasonable that middlebox-friendly schemes consider only a weaken adversary. In turn, this contradiction may be used as an argument against interception.

10.3 Middlebox privacy policy

While websites disclose their privacy policy to users, which may relate to e.g., data retention practices, middleboxes might follow different rules. This is particularly relevant when the middleboxes are not owned by either endpoints, e.g., for network performance enhancements. Privacy and legal issues may arise, especially in view of the European General Data Protection Regulation (GDPR) [87] and related privacy laws. We have not explicitly considered within this paper the privacy and legal issues; these appear to be open questions that further complicate potential deployability. Note that these issues are seldom discussed in the literature and would appear to deserve, we suggest, more attention.

https://t.me/learningnets

10.4 Alternatives to interception

For certain use cases, there are research avenues and industrially-deployed solutions that aim at understanding parts of the content hidden with TLS without decryption. The key idea is that encrypted traffic leaks some information that enables an adversary to understand parts of its content [110]. For instance, the number of TCP packets exchanged, their size and pace, can be used to fingerprint which website or even which page is being visited [91]. Defenses against this fingerprinting typically incur significant overhead [112]. The traffic metadata can also be used to distinguish traffic from benign or malicious applications [6, 26], based on advertised and negotiated ciphersuites, TLS version and extensions. Anderson et al. [5] also showed that specific HTTP headers could be inferred from HTTPS traffic. Reed et al. [92] could identify the videos played on Netflix over HTTPS by analyzing the TCP/IP headers only. Thus, it appears that specific scenarios could be addressed by deployable alternative solutions to interception and delegation.

10.5 Other open issues

From our analysis, we highlight the following open questions and issues to consider.

(1) When a TLS connection is terminated at a CDN, load balancer or firewall, and may not even reach the intended end-server, can it be called end-to-end? In some cases, the CDN acts as a host provider and could be considered a legitimate end. Similarly, if an enterprise terminates incoming TLS connections at an edge node (as in ETS), should the user be given control of the traffic up to an application server?
(2) Proposals such as ME-TLS [70] and TwinPeaks [21] rely on identity-based encryption (IBE) to achieve efficient key negotiation between endpoints and middleboxes for IoT monitoring. While pushing for IBE as a replacement for the web PKI is unrealistic, there might be a use for it in the IoT ecosystem.
(3) TLS inspection may negatively impact protocol performance improvements such as TLS session resumption [23] and 0-RTT. Moreover, performance-related use cases for access to plaintext traffic may not even benefit from traffic inspection proposals due to computational costs. In such a case, the middlebox could be given the information it seeks (e.g., preferred bitrate, type of content) as part of cleartext metadata that accompanies the TLS traffic, as done in EFGH; unfortunately, EFGH itself suffers from deployment challenges.
(4) Emerging countries and rural areas often suffer from poor connectivity/speed, low coverage and data quotas, making optimizations important. However, if supporting performance-enhancing proxies require changes in the HTTPS ecosystem, they will likely not be supported by all relevant stakeholders (e.g., content providers). Furthermore, the practice of customizing browsers for satellite-based ISP customers [67] is suboptimal.

10.6 Concluding remarks

Each common network practice for middleboxes that relies on access to plaintext TLS traffic comes with different requirements and implications. For the use case for Content Delivery Networks, which includes content caching and DDoS attack prevention, from our analysis it would appear that there are technically mature solutions, albeit not widely deployed, such as delegated credentials, proxy certificates, or trusted hardware-based solutions; for example, CloudFlare is pursuing delegated credentials [102] as discussed. For most other use cases, despite abundant enthusiasm in the literature, the existence of practical, deployable solutions has yet to be demonstrated; from our analysis, we expect that deployment barriers may be too hard to overcome. In particular, we have identified important open issues surrounding middlebox-friendly proposals including an increased complexity, expected end-user issues in transparency and understanding, and the lack of support for existing TLS performance optimization mechanisms. Also, it appears that industry arguments for plaintext access for compliance purposes are not explicitly supported by accompanying evidence (see our analysis of PCI-DSS and NERC CIP in §4.2).

https://t.me/learningnets

Importantly, an interesting question that appears not to have been seriously explored is what alternatives to TLS interception exist to address individual use cases, and which better align with everyone's incentives. Moreover, while there is a typical stakeholder conflict for operations motivated by law enforcement or regulatory controls enforced at the ISP, other conflicts such as the use of performance-enhancing middleboxes in high latency/low speed scenarios (e.g., rural areas or otherwise poorly served in Internet services) highlight difficult tradeoffs. Ultimately, given the importance of TLS—and it would appear that deployments and major service providers including Google are leaning towards HTTPS being the default replacing HTTP [99]—existing common network practices that rely on plaintext will apparently stop working or lose functionality. We encourage the community to explicitly discuss and reach consensus on the requirements for acceptable middlebox solutions, e.g., as an informational IETF RFC or working group draft, in particular considering specific use cases. This would ideally trigger discussion between industry and academic experts to agree on common grounds and focus the research effort to address specific issues. For example, one would expect that these requirements would include aspects such as rigorous independent protocol analysis before deployment (as with TLS 1.3), access to plaintext consistent with least privilege principles and privacy considerations, and accountability for any modification of content. A recent effort by public and private sector enterprises [19] helps to document a few of their requirements, e.g., scalability, effectiveness, but appears to prioritize access at the expense of the goals of privacy enthusiasts. We encourage similar effort but with a broader spectrum of stakeholders.

ACKNOWLEDGMENTS

The first author acknowledges funding from a DND/NSERC Discovery Grant supplement. The second author acknowledges funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) for both his Canada Research Chair in Authentication and Computer Security, and a Discovery Grant.

REFERENCES

[1] Victor Agababov, Michael Buettner, Victor Chudnovsky, Mark Cogan, Ben Greenstein, Shane McDaniel, Michael Piatek, Colin Scott, Matt Welsh, and Bolian Yin. 2015. Flywheel: Google's Data Compression Proxy for the Mobile Web. In *NSDI*.

[2] Devdatta Akhawe, Frederik Braun, François Marier, and Joel Weinberger. 2016. Subresource Integrity. https://www.w3.org/TR/SRI/.

[3] Abdulrahman Al-Dailami, Chang Ruan, Zhihong Bao, and Tao Zhang. 2019. QoS3: Secure Caching in HTTPS Based on Fine-Grained Trust Delegation. *Security and Communication Networks* (2019).

[4] Amazon Elastic Container Service. 2020. Load Balancer Types. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/load-balancer-types.html.

[5] Blake Anderson, Andrew Chi, Scott Dunlop, and David A. McGrew. 2019. Limitless HTTP in an HTTPS World: Inferring the Semantics of the HTTPS Protocol without Decryption. In *CODASPY*.

[6] Blake Anderson, Subharthi Paul, and David A. McGrew. 2018. Deciphering malware's use of TLS (without decryption). *Journal of Computer Virology and Hacking Techniques* 14, 3 (2018), 195–211.

[7] Flemming Andreasen, Nancy Cam-Winget, and Eric Wang. 2019. TLS 1.3 Impact on Network-Based Security. Internet Draft (Informational).

[8] Australian Governement. 2015. Telecommunications (Interception and Access) Amendment (Data Retention) Act 2015. https://www.legislation.gov.au/Details/C2015A00039.

[9] Fred Baker, Bill Foster, and Chip Sharp. 2004. Cisco Architecture for Lawful Intercept in IP Networks. RFC 3924 (Informational Track).

[10] Richard Barnes, Subodh Iyengar, Nick Sullivan, and Eric Rescorla. 2020. Delegated Credentials for TLS. Internet Draft (Standards Track). https://tools.ietf.org/html/draft-ietf-tls-subcerts-09.

[11] Karthikeyan Bhargavan, Ioana Boureanu, Antoine Delignat-Lavaud, Pierre-Alain Fouque, and Cristina Onete. 2018. A Formal Treatment of Accountable Proxying Over TLS. In *IEEE S&P*.

[12] Karthikeyan Bhargavan, Ioana Boureanu, Pierre-Alain Fouque, Cristina Onete, and Benjamin Richard. 2017. Content delivery over TLS: a cryptographic analysis of Keyless SSL. In *EuroS&P*.

[13] Michael Bierma, Aaron Brown, Troy DeLano, Thomas M. Kroeger, and Howard Poston. 2017. Locally Operated Cooperative Key Sharing (LOCKS). In *International Conference on Computing, Networking and Communications (ICNC'17)*.

[14] BITS Security. 2016. [TLS] Industry Concerns about TLS 1.3. IETF mail (Sep. 22, 2016). https://mailarchive.ietf.org/arch/msg/tls/KQIyNhPk8K6jOoe2ScdPZ8E08RE/.

https://t.me/learningnets

[15] Colin Boyd, Anish Mathuria, and Douglas Stebila. 2020. *Protocols for Authentication and Key Establishment, Second Edition.* Springer. https://doi.org/10.1007/978-3-662-58146-9
[16] Jon Brodkin. 2016. Verizon's "supercookies" violated net neutrality transparency rule. https://arstechnica.com/information-technology/2016/03/verizons-supercookies-violated-net-neutrality-transparency-rule/.
[17] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *USENIX Security*.
[18] Sonia Burney. 2018. Security and Mutual SSL Identity Authentication for IoT. Akamai white paper. https://www.akamai.com/us/en/multimedia/documents/white-paper/akamai-security-and-mutual-ssl-iot-white-paper.pdf.
[19] Center for Cybersecurity Policy and Law. 2019. Enterprise Data Center Transparency and Security Initiative. https://centerforcybersecuritypolicy.org/enterprise-data-center-transparency-and-security-initiative.
[20] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten-Hwang Lai. 2019. SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution. In *EuroS&P*.
[21] Eunsang Cho, Jeongnyeo Kim, Minkyung Park, Hyeonmin Lee, Chorom Hamm, Soobin Park, Sungmin Sohn, Minhyeok Kang, and Ted Taekyoung Kwon. 2020. TwinPeaks: An approach for certificateless public key distribution for the internet and internet of things. *Computer Networks* 175 (2020), 107268.
[22] David Choffnes, Phillipa Gill, and Alan Mislove. 2017. An Empirical Evaluation of Deployed DPI Middleboxes and Their Implications for Policymakers. In *Research Conference on Communications, Information, and Internet Policy (TPRC)*.
[23] Laurent Chuat, AbdelRahman Abdou, Ralf Sasse, Christoph Sprenger, David Basin, and Adrian Perrig. 2020. SoK: Delegation and Revocation, the Missing Links in the Web's Chain of Trust. In *EuroS&P*.
[24] Taejoong Chung, David Choffnes, and Alan Mislove. 2016. Tunneling for Transparency: A Large-Scale Analysis of End-to-End Violations in the Internet. In *IMC*.
[25] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David R. Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. 2017. A Longitudinal, End-to-End View of the DNSSEC Ecosystem. In *USENIX Security*.
[26] Cisco Systems, Inc. 2019. Cisco Encrypted Traffic Analytics. White paper. https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf.
[27] Cisco Systems, Inc. 2020. ECS Administration Guide, StarOS Release 21.18. https://www.cisco.com/c/en/us/td/docs/wireless/asr_5000/21-18_6-12/ECS-Admin/21-18-ECS-Admin.pdf.
[28] Jeremy Clark and Paul C. van Oorschot. 2013. SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. In *IEEE S&P*.
[29] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and William Polk. 2008. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. RFC 5280 (Standards Track).
[30] Cornell University. [n.d.]. Data Transfer: Globus Toolkit and GSI. https://cvw.cac.cornell.edu/datatransfer/gsi.
[31] Court of Justice of the European Union. 2014. The Court of Justice declares the Data Retention Directive to be invalid. https://curia.europa.eu/jcms/upload/docs/application/pdf/2014-04/cp140054en.pdf.
[32] Xavier de Carné de Carnavalet and Mohammad Mannan. 2016. Killed by Proxy: Analyzing Client-end TLS Interception Software. In *NDSS*.
[33] Xavier de Carné de Carnavalet and Mohammad Mannan. 2019. Privacy and Security Risks of "Not-a-Virus" Bundled Adware: The Wajam Case. *CoRR* abs/1905.05224 (2019).
[34] Tim Dierks and Christopher Allen. 1999. The TLS Protocol Version 1.0. RFC 2246 (Standards Track).
[35] Tim Dierks and Eric Rescorla. 2006. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Standards Track).
[36] Tim Dierks and Eric Rescorla. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Standards Track).
[37] Viktor Dukhovni and Wes Hardaker. 2015. The DNS-Based Authentication of Named Entities (DANE) Protocol: Updates and Operational Guidance. RFC 7671 (Standards Track).
[38] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J Alex Halderman, and Vern Paxson. 2017. The Security Impact of HTTPS Interception. In *NDSS*.
[39] ETSI. 2006. Lawful Interception (LI) – Interception domain Architecture for IP networks. ETSI TR 102 528 V1.1.1.
[40] ETSI. 2019. Middlebox Security Protocol – Part 2: Transport layer MSP, profile for fine grained access control. ETSI TS 103 523-2 V0.1.0. https://portal.etsi.org/webapp/workProgram/Report_WorkItem.asp?wki_id=52930.
[41] ETSI. 2019. Middlebox Security Protocol – Part 3: Enterprise Transport Security. ETSI TS 103 523-3 V1.3.1.
[42] Chris Evans, Chris Palmer, and Ryan Sleevi. 2015. Public Key Pinning Extension for HTTP. RFC 7469 (Standards Track).
[43] Exégètes Amateurs. 2018. Application of the Tele2 Sverige/Watson jurisprudence throughout Europe. Open letter to the European Commission. https://stopdataretention.eu/.
[44] Stephen Farrell. 2017. tinfoil: TLS Is Not For Obligatory (Or Ostensibly Optional) Interception, Luckily. https://github.com/sftcd/tinfoil.

https://t.me/learningnets

[45] Thomas Fossati, Vijay K. Gurbani, and Vladimir Kolesnikov. 2015. Love All, Trust Few: on Trusting Intermediaries in HTTP. In *SIGCOMM HotMiddlebox*.
[46] Nikos Fotiou, Vasilios A. Siris, Mario Marchese, Franco Davoli, Luca Boero, and George C. Polyzos. 2019. Exploiting Satellite Broadcast Despite HTTPS. In *IEEE Global Communications Conference (GLOBECOM)*.
[47] Alan O. Freier, Philip Karlton, and Paul C. Kocher. 2011. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic).
[48] Charles E Gero and Michael R Stone. 2015. Splicing into an active TLS session without a certificate or private key. US Patent 9,137,218.
[49] Daniel K. Gillmor. 2018. TLS clients should reject static Diffie-Hellman. Internet Draft (Standards Track). https://tools.ietf.org/id/draft-dkg-tls-reject-static-dh-00.html.
[50] Dan Goodin. 2015. Google Chrome will banish Chinese certificate authority for breach of trust. https://arstechnica.com/security/2015/04/google-chrome-will-banish-chinese-certificate-authority-for-breach-of-trust/.
[51] Dan Goodin. 2017. Nope, this isn't the HTTPS-validated Stripe website you think it is. https://arstechnica.com/information-technology/2017/12/nope-this-isnt-the-https-validated-stripe-website-you-think-it-is/.
[52] M. Green, R. Droms, R. Housley, P. Turner, and S. Fenter. 2017. Data Center use of Static Diffie-Hellman in TLS 1.3. Internet Draft (Standards Track). https://tools.ietf.org/html/draft-green-tls-static-dh-in-tls13-01.
[53] Glenn Greenwald and Ewen MacAskill. 2013. NSA Prism program taps in to user data of Apple, Google and others. https://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data.
[54] Juhyeng Han, Seong Min Kim, Jaehyeong Ha, and Dongsu Han. 2017. SGX-Box: Enabling Visibility on Encrypted Traffic using a Secure Middlebox Module. In *Asia-Pacific Workshop on Networking (APNet'17)*.
[55] Stephen Herwig, Christina Garman, and Dave Levin. 2020. Achieving Keyless CDNs with Conclaves. In *USENIX Security*.
[56] Jacob Hoffman-Andrews. 2019. ETS Isn't TLS and You Shouldn't Use It. https://www.eff.org/deeplinks/2019/02/ets-isnt-tls-and-you-shouldnt-use-it.
[57] Johann Hofmann. 2019. Intent to Ship: Move Extended Validation Information out of the URL bar. Google Groups firefox-dev forum (Aug. 12, 2019). https://groups.google.com/forum/m/?fromgroups&hl=en#!topic/firefox-dev/6wAg_PpnlY4.
[58] Russ Housley and Ralph Droms. 2018. TLS 1.3 Option for Negotiation of Visibility in the Datacenter. Internet Draft (Standards Track). https://tools.ietf.org/html/draft-rhrd-tls-tls13-visibility-01.
[59] Internet Society. 2017. Perspectives on Internet Content Blocking: An Overview. https://www.internetsociety.org/wp-content/uploads/2017/03/ContentBlockingOverview.pdf.
[60] Jeff Jarmoc. 2012. SSL Interception Proxies and Transitive Trust. In *BlackHat Europe*.
[61] Sheharbano Khattak, David Fifield, Sadia Afroz, Mobin Javed, Srikanth Sundaresan, Damon McCoy, Vern Paxson, and Steven J. Murdoch. 2016. Do You See What I See? Differential Treatment of Anonymous Users. In *NDSS*.
[62] Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G. Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. 2018. Coming of Age: A Longitudinal Study of TLS Deployment. In *IMC*.
[63] Hugo Krawczyk and Pasi Eronen. 2010. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (Informational Track).
[64] Chang Lan, Justine Sherry, Raluca Ada Popa, Sylvia Ratnasamy, and Zhi Liu. 2016. Embark: Securely Outsourcing Middleboxes to the Cloud. In *NSDI*.
[65] Eric Lawrence. 2019. Spying on HTTPS. https://textslashplain.com/2019/08/11/spying-on-https/.
[66] Hyunwoo Lee, Zach Smith, Junghwan Lim, Gyeongjae Choi, Selin Chun, Taejoong Chung, and Ted Taekyoung Kwon. 2019. maTLS: How to Make TLS middlebox-aware?. In *NDSS*.
[67] Peter Lepeska. 2014. Trusted Proxy and the Cost of Bits. IETF HTTP (httpbis) WG meeting slides. https://datatracker.ietf.org/doc/slides-90-httpbis-6/.
[68] Chris Lesniewski-Laas and M. Frans Kaashoek. 2005. SSL splitting: Securely serving data from untrusted caches. *Computer Networks* 48, 5 (2005), 763–779.
[69] John Leyden. 2012. Revoking Trust in two TurkTrust Certificates. https://www.theregister.com/2012/02/14/trustwave_analysis/.
[70] Jie Li, Rongmao Chen, Jinshu Su, Xinyi Huang, and Xiaofeng Wang. 2020. ME-TLS: Middlebox-Enhanced TLS for Internet-of-Things Devices. *IEEE Internet of Things Journal* 7, 2 (2020), 1216–1229.
[71] Jinjin Liang, Jian Jiang, Haixin Duan, Kang Li, Tao Wan, and Jianping Wu. 2014. When HTTPS Meets CDN: A Case of Authentication in Delegated Service. In *USENIX Security*.
[72] David A. McGrew, Dan Wing, Yoav Nir, and Philip Gladstone. 2012. TLS Proxy Server Extension. Internet Draft (Informational). https://tools.ietf.org/html/draft-mcgrew-tls-proxy-server-01.
[73] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. 2013. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Computer Aided Verification (CAV)*.
[74] Daniel Migault. 2018. LURK Extension version 1 for (D)TLS 1.2 Authentication. Internet Draft (Standards Track).
[75] Daniel Migault. 2018. LURK Protocol version 1. Internet Draft (Standards Track).
[76] Daniel Migault. 2019. LURK Extension version 1 for (D)TLS 1.3 Authentication. Internet Draft (Standards Track).

https://t.me/learningnets

[77] Kathleen Moriarty and Al Morton. 2018. Effects of Pervasive Encryption on Operators. RFC 8404 (Informational Track).
[78] Mozilla. 2013. Revoking Trust in one ANSSI Certificate. https://blog.mozilla.org/security/2013/12/09/revoking-trust-in-one-anssi-certificate/.
[79] Mozilla. 2013. Revoking Trust in two TurkTrust Certificates. https://blog.mozilla.org/security/2013/01/03/revoking-trust-in-two-turktrust-certficates/.
[80] Mozilla. 2019. NSS Key Log Format. https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format.
[81] Hyunwoo Nam, Kyung-Hwa Kim, Bong-Ho Kim, Doru Calin, and Henning Schulzrinne. 2014. Towards dynamic QoS-aware over-the-top video streaming. In *WoWMoM*.
[82] David Naylor, Richard Li, Christos Gkantsidis, Thomas Karagiannis, and Peter Steenkiste. 2017. And Then There Were More: Secure Communication for More Than Two Parties. In *CoNEXT*.
[83] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R. López, Konstantina Papagiannaki, Pablo Rodríguez Rodríguez, and Peter Steenkiste. 2015. Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS. In *SIGCOMM*.
[84] Jianting Ning, Geong Sen Poh, Jia-Ch'ng Loh, Jason Chia, and Ee-Chien Chang. 2019. PrivDPI: Privacy-Preserving Encrypted Traffic Inspection with Reusable Obfuscated Rules. In *CCS*.
[85] Nubeva. 2019. Decrypted Visibility in a TLS 1.3 World – Symmetric Key Intercept for Secure SSL / TLS Decryption in the Cloud. White Paper. https://info.nubeva.com/hubfs/Downloadables/Nubeva%20Decrypted%20Visibility%20-%20white%20paper%20-%20v4.pdf.
[86] Devon O'Brien. 2019. Upcoming Change to Chrome's Identity Indicators. Google Groups chromium.org Security-dev's list (Aug. 8, 2019). https://groups.google.com/a/chromium.org/forum/#!topic/security-dev/h1bTcoTpfeI.
[87] Official Journal of the European Union. 2016. General Data Protection Regulation. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46. 59 (2016), 1–88.
[88] OpenSSL. 2005. Changelog. https://www.openssl.org/news/changelog.html.
[89] Opera. 2015. Choose High or Extreme savings in new Opera Mini for Android. https://blogs.opera.com/mobile/2015/09/choose-high-or-extreme-savings-new-opera-mini-android/.
[90] Tavis Ormandy. 2016. Kaspersky: SSL interception differentiates certificates with a 32bit hash. Google Project Zero Issue #978 (Nov. 1, 2016). https://bugs.chromium.org/p/project-zero/issues/detail?id=978.
[91] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website Fingerprinting at Internet Scale. In *NDSS*.
[92] Andrew Reed and Michael Kranch. 2017. Identifying HTTPS-Protected Netflix Videos in Real-Time. In *CODASPY*.
[93] Jingjing Ren, Daniel J. Dubois, David R. Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. 2019. Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach. In *IMC*.
[94] Eric Rescorla. 2001. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley.
[95] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Standards Track).
[96] Eric Rescorla, Marsh Ray, Steve Dispensa, and Nasko Oskov. 2010. Transport Layer Security (TLS) Renegotiation Indication Extension. RFC 5746 (Standards Track).
[97] Ivan Ristić. 2014. *Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications*. Feisty Duck. Revised in July 2017, build 821.
[98] Charles H Romine. 2018. Public Comments on the Second Draft of NIST Special Publication 800-52 Revision 2, Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations. Oct. 15, 2018. https://csrc.nist.gov/CSRC/media/Publications/sp/800-52/rev-2/draft/documents/sp800-52r2-draft2-comments-received.pdf.
[99] Emily Schechter. 2018. A milestone for Chrome security: marking HTTP as "not secure". https://blog.google/products/chrome/milestone-chrome-security-marking-http-not-secure/.
[100] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. 2015. BlindBox: Deep Packet Inspection over Encrypted Traffic. In *SIGCOMM*.
[101] Nick Sullivan. 2014. Keyless SSL: The Nitty Gritty Technical Details. https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details/.
[102] Nick Sullivan and Watson Ladd. 2019. Delegated Credentials for TLS. https://blog.cloudflare.com/keyless-delegation/.
[103] Willy Tarreau. 2010. The PROXY protocol. HAProxy load balancer feature. https://www.haproxy.org/download/1.8/doc/proxy-protocol.txt.
[104] Roelof Du Toit. 2017. Responsibly Intercepting TLS and the Impact of TLS 1.3. Symantec white paper. https://docs.broadcom.com/doc/responsibly-intercepting-tls-and-the-impact-of-tls-1.3.en.
[105] Steven Tuecke, Von Welch, Doug Engert, Laura Pearlman, and Mary Thompson. 2004. Internet X.509 Public Key Infrastructure (PKI). RFC 3820 (Standards Track).

https://t.me/learningnets

[106] Daiki Ueno. 2020. Understanding the DTLS all-zero ClientHello.random vulnerability. Red Hat blog article (May 13, 2020). https://www.redhat.com/en/blog/understanding-dtls-all-zero-clienthellorandom-vulnerability.
[107] Narseo Vallina-Rodriguez, Srikanth Sundaresan, Christian Kreibich, and Vern Paxson. 2015. Header Enrichment or ISP Enrichment: Emerging Privacy Threats in Mobile Networks. In *SIGCOMM HotMiddlebox*.
[108] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lipp, Marina Minkin, Daniel Genkin, Yarom Yuval, Berk Sunar, Daniel Gruss, and Frank Piessens. 2020. LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection. In *IEEE S&P*.
[109] Steven J. Vaughan-Nichols. 2013. Google, the NSA, and the need for locking down datacenter traffic. News article (Oct. 30, 2013). https://www.zdnet.com/article/google-the-nsa-and-the-need-for-locking-down-datacenter-traffic/.
[110] Petr Velan, Milan Cermák, Pavel Celeda, and Martin Drasar. 2015. A survey of methods for encrypted traffic classification and analysis. *Int. Journal of Network Management* 25, 5 (2015), 355–374.
[111] Louis Waked, Mohammad Mannan, and Amr M. Youssef. 2018. To Intercept or Not to Intercept: Analyzing TLS Interception in Network Appliances. In *ASIACCS*.
[112] Tao Wang and Ian Goldberg. 2017. Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks. In *USENIX Security*.
[113] Alec Waters. 2019. Eyesight to the Blind - SSL Decryption for Network Monitoring. https://resources.infosecinstitute.com/ssl-decryption/.
[114] Nicholas Weaver, Christian Kreibich, Martin Dam, and Vern Paxson. 2014. Here Be Web Proxies. In *Passive and Active Measurement (PAM)*.
[115] Changzheng Wei, Jian Li, Weigang Li, Ping Yu, and Haibing Guan. 2017. STYX: a trusted and accelerated hierarchical SSL key management and distribution system for cloud based CDN application. In *Symposium on Cloud Computing (SoCC)*.
[116] Von Welch, Ian Foster, Carl Kesselman, Olle Mulmo, Laura Pearlman, Steven Tuecke, Jarek Gawor, Sam Meder, and Frank Siebenlist. 2004. X.509 proxy certificates for dynamic delegation. In *3rd Annual NIST PKI R&D workshop*. Gaithersburg, MD, USA.
[117] Judson Wilson, Riad S. Wahby, Henry Corrigan-Gibbs, Dan Boneh, Philip Levis, and Keith Winstein. 2017. Trust but Verify: Auditing the Secure Internet of Things. In *MobiSys*.
[118] Zhenyu Zhou and Theophilus Benson. 2015. Towards a Safe Playground for HTTPS and Middle Boxes with QoS2. In *SIGCOMM HotMiddlebox*.

A AKAMAI'S SPLICING PATENT CAPABILITY REVOCATION ISSUES

This appendix details an apparent flaw in the Akamai patent [48] discussed in §6.1.1.

When the edge server possesses the `client_` and `server_write_key` (after it was allowed to observe client requests and serve a cached request), it can impersonate the client and perform a renegotiation with the origin server without the client being involved. The server would then believe that the client successfully renegotiated and the edge server is unable to read further, while the edge server actually became the client in the communication. The edge server can then still decrypt client requests (with the former `client_write_key`), forward them or forge requests to the origin server and decrypt the response. However, it would not be able to forward the response back to the client, which limits the practicality of this attack.

The impersonation of the renegotiation request is possible before TLS 1.3 as all necessary keys are known to the edge server. Even when using the secure renegotiation extension [96], the edge server is able to compute the required `client_verify_data`, based on the decrypted two `verify_data` in the `Finished` messages from the client and the server (encrypted with the `client_write_key` and `server_write_key` in TLS 1.2, respectively). In TLS 1.3, the encrypted handshake messages are protected with different keys than application messages, which would defeat this attack. However, in this version of TLS, non-AEAD ciphersuites have been removed, making it incompatible with Akamai's patent.

https://t.me/learningnets