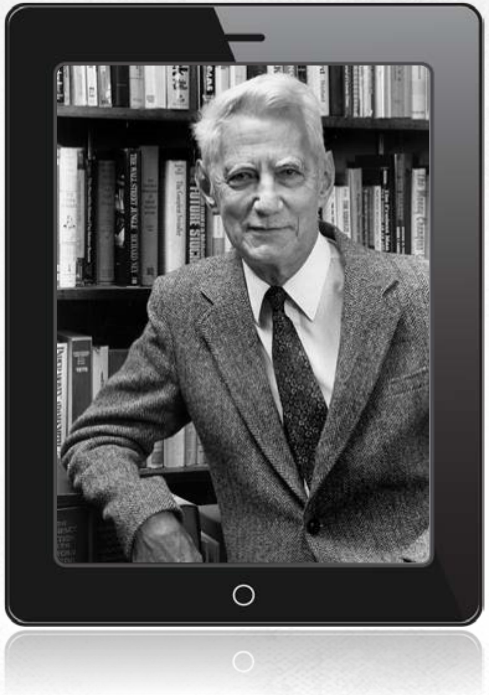
# Symmetric and Asymmetric Ciphers: how they work and what they are for

# Information Theory

Information theory is a field dealing with information processing methods. Started by Claude E. Shannon in *A Mathematical Theory of Cryptography* and *A Mathematical Theory of Communication*



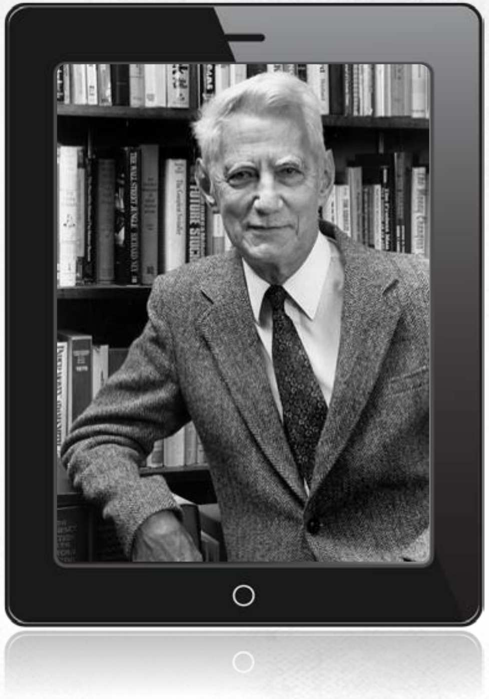
# Information Theory

Bit: the smallest unit of information needed to encode which of two possibilities happened

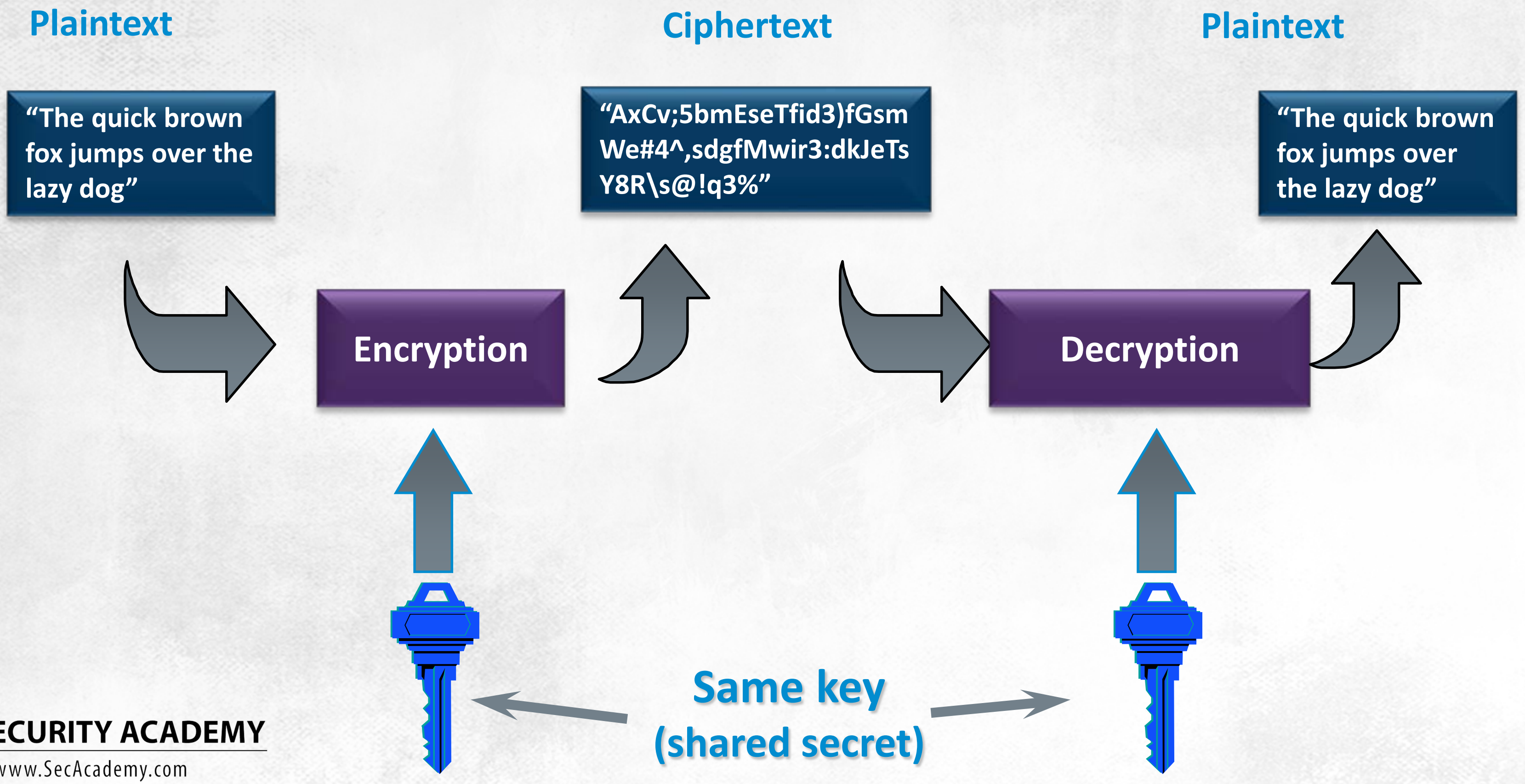
Entropy: the smallest amount of information needed, on the average, to encode an event out of a set of events at a given level of probability

Confusion: hiding the relationships between the plaintext, the ciphertext and the key. Confusion is measured by how probable linear approximation is

Diffusion: the distribution of plaintext and key bits over the ciphertext. Diffusion is measured by the probability of a successful differential approximation



# Symmetric-key Algorithms


Plaintext
Ciphertext
Plaintext
"The quick brown fox jumps over the lazy dog"
"AxCv;5bmEseTfid3)fGsm We#4^,sdgfMwir3:dkJeTs Y8R\s@!q3%"
"The quick brown fox jumps over the lazy dog"
Encryption
Decryption
Same key (shared secret)

# Symmetric-key algorithms

Pros:

- Speed (1,000 to 10,000 times faster than asymmetric-key algorithms)
- Easy and cheap hardware implementation

Cons:

- Key needs to be negotiated
  - If attackers obtain the key, this allows them to both decrypt the messages and also modify it and encrypt it again in a transparent way
- Enormous number of keys: n(n–1)/2, where n is the number of system users (1,225 for n=50)
  - If the sender uses the same key to encrypt messages intended for multiple recipients, each of the recipients can decrypt the messages intended for other system users

## Symmetric-key algorithms

Characteristic features:

- Rounds (by repeating a weak operation you make an attack harder to pull off)
- Round keys

To ensure confusion and diffusion are adequate, symmetric-key algorithms should:

- Distribute plaintext bits randomly and evenly over the ciphertext, or mix the ciphertext
- Diffuse adjacent plaintext bits so that each has a bearing on many (ideally all) ciphertext bits.
- Another plus of this spreading is that changing one bit in the message will change many bits found in many places in the ciphertext

Depending on the number of data encrypted at a time, symmetric-key algorithms can be divided into:

- Block ciphers, which encrypt and decrypt a set of information (a block of bits) at a time
- Stream ciphers, which encrypt and decrypt a unit of information (bit) at the same time

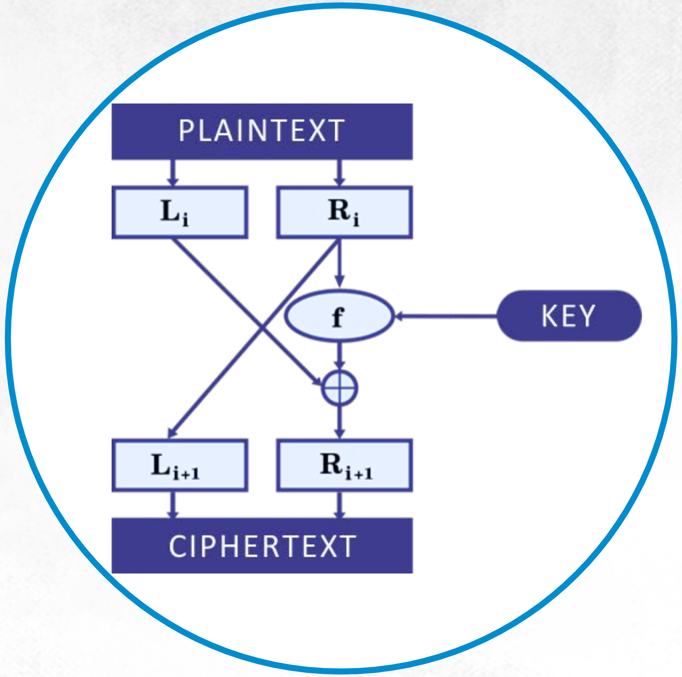
# Block CIPHERS

## Feistel's Network

Most symmetric ciphers are block ciphers

Block cipher mode defines the transformation method of variable-length messages (streams of data) into fixed-length blocks

Most block ciphers use Feistel's network, which may be used with any encryption function and doesn't require the function to be reversible



# Block CIPHERS

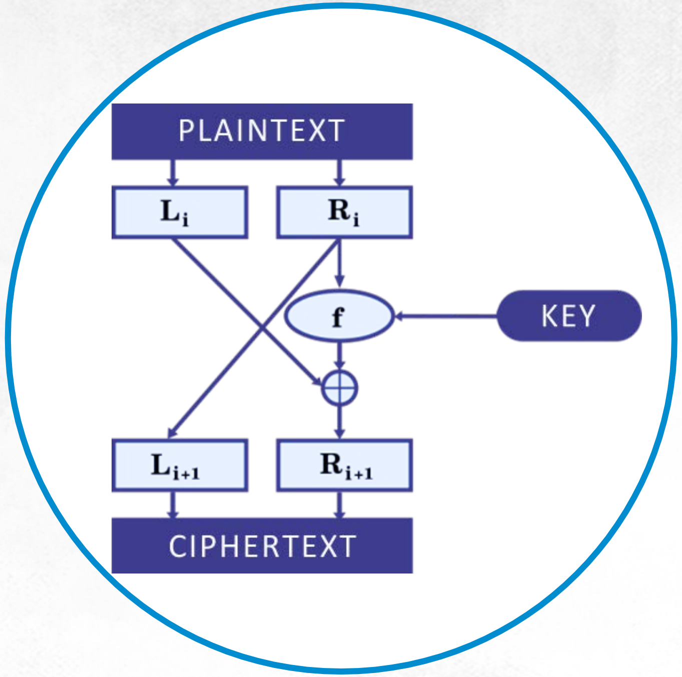
## Feistel's Network

Encryption rounds look like this:

- A data block is split into two halves, $L_0$ and $R_0$
- One half is encrypted with the function f using the round key $K_0$
- New data halves are computed, and $R_0$ becomes $L_1$, while the output of encrypting $R_0$ combined with $L_0$ using XOR becomes $R_1$

Decryption is the reversal of splitting data in half:

- $R_{n-1} = L_n$
- $L_{n-1} = f(K_n,R_{n-1})$ XOR $R_n = f(K_n,L_n)$ XOR $R_n$

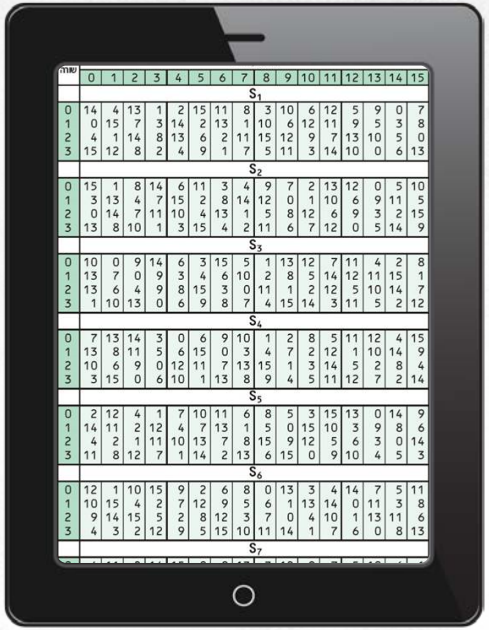


# Block Ciphers

## Feistel's Network

The security of the ciphertext emerging from Feistel's network rests solely on the function f (the type of symmetric encryption used). The function f is charged with four tasks:

- Mixing bits to obtain a non-linear pseudo-random relationship between ciphertext and message
- Perform bit permutation to flip the order of bits
- Diffuse bits to get the avalanche effect (the change of one plaintext bit results in the change of many bits in the ciphertext)
- Compress ciphertext to make its length and plaintext length match



| שורה | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $S_1$ | | | | | | | | | | | | | | | |
| 0 | 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| 1 | 0 | 15 | 7 | 3 | 14 | 2 | 13 | 1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| 2 | 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 | 15 | 12 | 9 | 7 | 13 | 10 | 5 | 0 |
| 3 | 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |
| | $S_2$ | | | | | | | | | | | | | | | |
| 0 | 15 | 1 | 8 | 14 | 6 | 11 | 3 | 4 | 9 | 7 | 2 | 13 | 12 | 0 | 5 | 10 |
| 1 | 3 | 13 | 4 | 7 | 15 | 2 | 8 | 14 | 12 | 0 | 1 | 10 | 6 | 9 | 11 | 5 |
| 2 | 0 | 14 | 7 | 11 | 10 | 4 | 13 | 1 | 5 | 8 | 12 | 6 | 9 | 3 | 2 | 15 |
| 3 | 13 | 8 | 10 | 1 | 3 | 15 | 4 | 2 | 11 | 6 | 7 | 12 | 0 | 5 | 14 | 9 |
| | $S_3$ | | | | | | | | | | | | | | | |
| 0 | 10 | 0 | 9 | 14 | 6 | 3 | 15 | 5 | 1 | 13 | 12 | 7 | 11 | 4 | 2 | 8 |
| 1 | 13 | 7 | 0 | 9 | 3 | 4 | 6 | 10 | 2 | 8 | 5 | 14 | 12 | 11 | 15 | 1 |
| 2 | 13 | 6 | 4 | 9 | 8 | 15 | 3 | 0 | 11 | 1 | 2 | 12 | 5 | 10 | 14 | 7 |
| 3 | 1 | 10 | 13 | 0 | 6 | 9 | 8 | 7 | 4 | 15 | 14 | 3 | 11 | 5 | 2 | 12 |
| | $S_4$ | | | | | | | | | | | | | | | |
| 0 | 7 | 13 | 14 | 3 | 0 | 6 | 9 | 10 | 1 | 2 | 8 | 5 | 11 | 12 | 4 | 15 |
| 1 | 13 | 8 | 11 | 5 | 6 | 15 | 0 | 3 | 4 | 7 | 2 | 12 | 1 | 10 | 14 | 9 |
| 2 | 10 | 6 | 9 | 0 | 12 | 11 | 7 | 13 | 15 | 1 | 3 | 14 | 5 | 2 | 8 | 4 |
| 3 | 3 | 15 | 0 | 6 | 10 | 1 | 13 | 8 | 9 | 4 | 5 | 11 | 12 | 7 | 2 | 14 |
| | $S_5$ | | | | | | | | | | | | | | | |
| 0 | 2 | 12 | 4 | 1 | 7 | 10 | 11 | 6 | 8 | 5 | 3 | 15 | 13 | 0 | 14 | 9 |
| 1 | 14 | 11 | 2 | 12 | 4 | 7 | 13 | 1 | 5 | 0 | 15 | 10 | 3 | 9 | 8 | 6 |
| 2 | 4 | 2 | 1 | 11 | 10 | 13 | 7 | 8 | 15 | 9 | 12 | 5 | 6 | 3 | 0 | 14 |
| 3 | 11 | 8 | 12 | 7 | 1 | 14 | 2 | 13 | 6 | 15 | 0 | 9 | 10 | 4 | 5 | 3 |
| | $S_6$ | | | | | | | | | | | | | | | |
| 0 | 12 | 1 | 10 | 15 | 9 | 2 | 6 | 8 | 0 | 13 | 3 | 4 | 14 | 7 | 5 | 11 |
| 1 | 10 | 15 | 4 | 2 | 7 | 12 | 9 | 5 | 6 | 1 | 13 | 14 | 0 | 11 | 3 | 8 |
| 2 | 9 | 14 | 15 | 5 | 2 | 8 | 12 | 3 | 7 | 0 | 4 | 10 | 1 | 13 | 11 | 6 |
| 3 | 4 | 3 | 2 | 12 | 9 | 5 | 15 | 10 | 11 | 14 | 1 | 7 | 6 | 0 | 8 | 13 |
| | $S_7$ | | | | | | | | | | | | | | | |

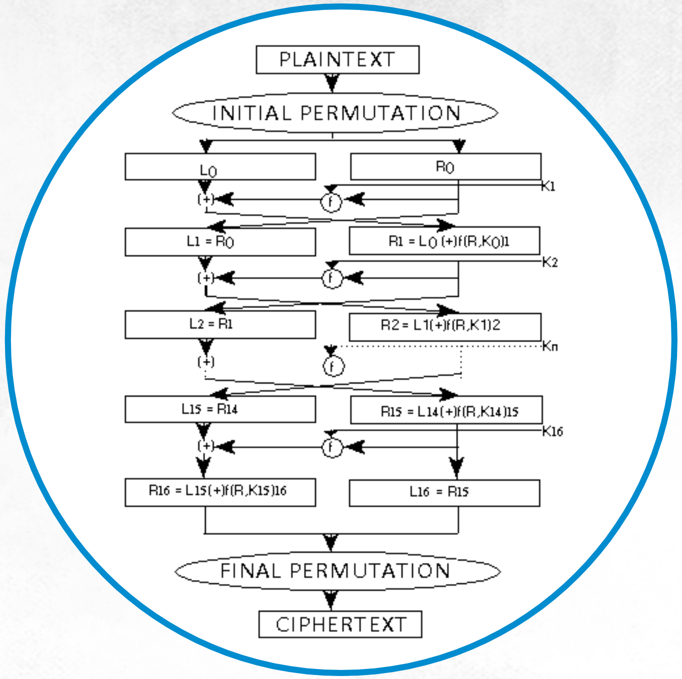
# Data Encryption standard

Developed by IBM in the 70s

Encrypts 64-bit data blocks (equivalent to 8 letters in the ASCII code plus a parity bit) using a 56-bit key

Encryption and decryption take 16 Feistel rounds

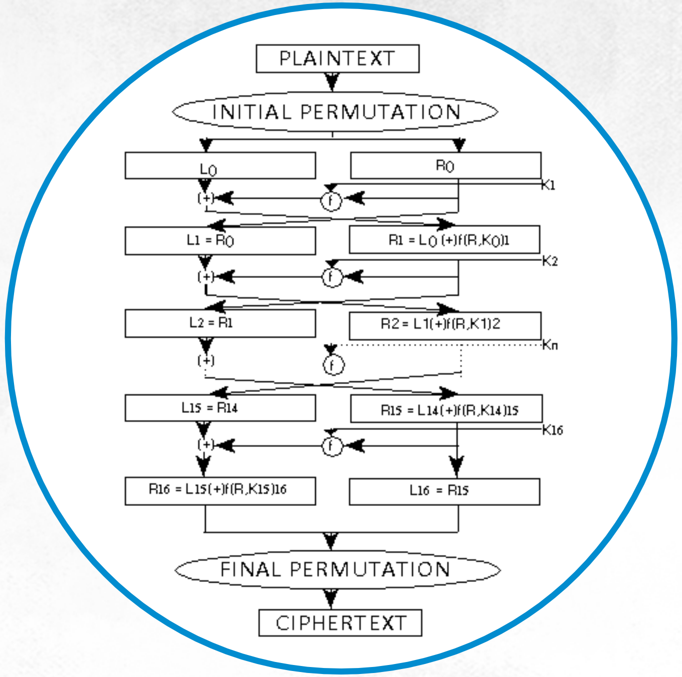
Prior to the first round a message block undergoes a permanent permutation, which is of no great consequence to the security of the ciphertext



# Data Encryption standard

In each round, 32-bit halves of the data block are expanded to 48 bits. The result of this expansion is combined using the XOR operation with a round key, and this value is split into six 8-bit parts, each of them is processed using a substitution-permutation network

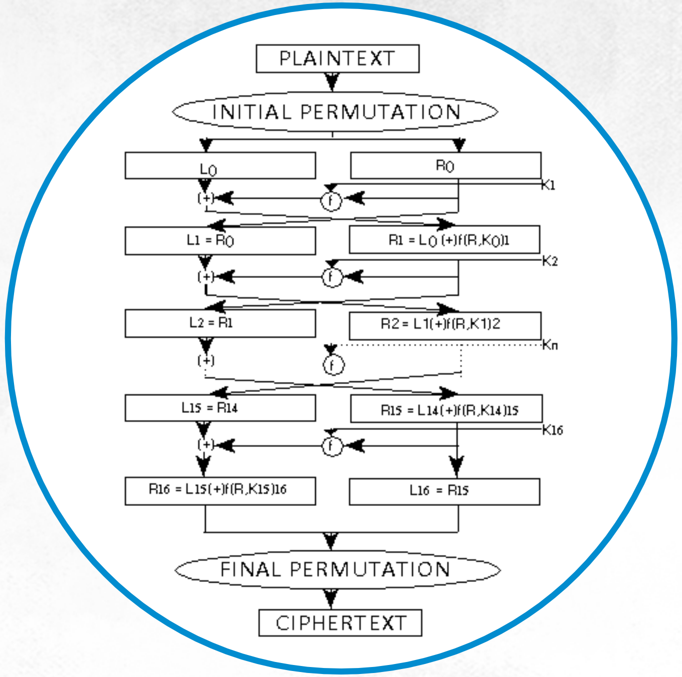
All networks are 4 x 16 tables, with rows containing all numbers from 0 to 15. The first and last bit of encrypted parts of data are used as row numbers, and the rest are used as column numbers. The numbers contained in a substitution-permutation network are the result of encryption



# Data Encryption standard

Combined outputs of the networks then undergo permanent permutation

Additionally, after the last round, the ciphertext block undergoes a permutation that is a reversal of the initial permutation

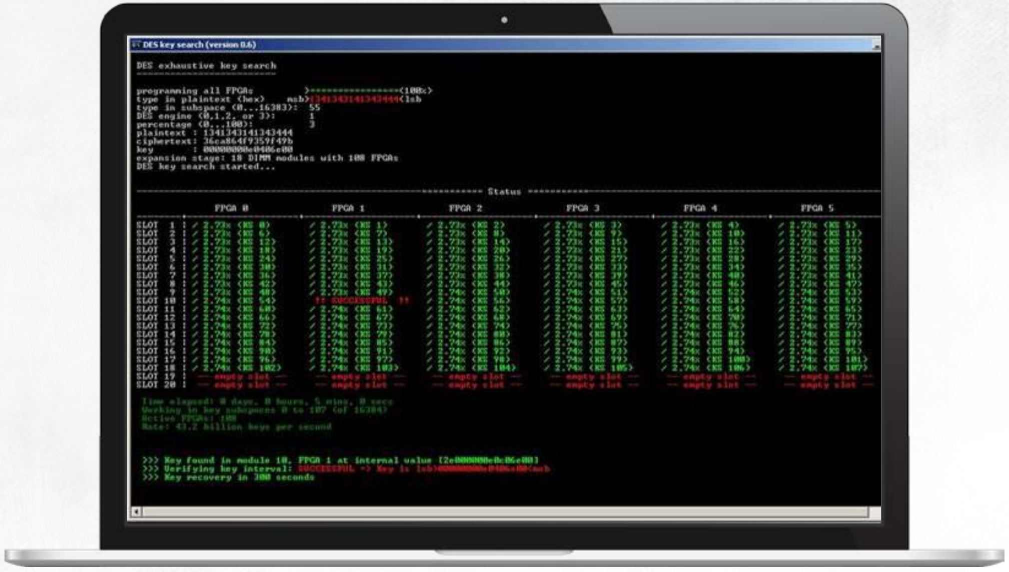


IT SECURITY ACADEMY
www.SecAcademy.com

# Data Encryption standard

DES weak points:

- Block size too small
- Key too short, offering a strength of 28 bits (half the key length). This allowed DES to be brute-forced as far back as 1999 in 22 hours
- Strong dependency of round keys on encryption keys, especially if the key is 0: round keys will also be 0
- Padding, or the interdependencies between keys, plaintexts and ciphertexts: when you encrypt the padding of all plaintext bits with the key padding you get the ciphertext padding



```
DES key search (version 0.6)

DES exhaustive key search
================================

programming all FPGAs:            >==================<100%>
type in plaintext (hex)      msb>1341343141343444<lsb
type in subspace (0...16383):   55
DES engine (0,1,2, or 3):        1
percentage (0...100):            3
plaintext : 1341343141343444
ciphertext: 36ca864f9359f49b
key       : 00000000e0406e00
expansion stage: 18 DIMM modules with 108 FPGAs
DES key search started...

------------------------------------------ Status ------------------------------------------

[illegible]

Time elapsed: 0 days, 0 hours, 5 mins, 0 secs
Working in key subspaces 0 to 107 (of 16384)
Active FPGAs: 108
Rate: 43.2 billion keys per second

>>> Key found in module 10, FPGA 1 at internal value [2e000000e0c06e00]
>>> Verifying key interval: SUCCESSFUL -> Key is lsb>[illegible]<msb
>>> Key recovery in 300 seconds
```

IT SECURITY ACADEMY
www.SecAcademy.com

## Data Encryption standard

Today DES may be cracked in just several minutes, so it cannot be used to protect the confidentiality of data

**In 2004 NIST recommended that the original DES algorithm should no longer be used**

# Des-X

DES-X is a DES modification presented by Ron Rivest in May 1984

The goal of this update was to lengthen the key without changing how the algorithm operates. This aimed to make brute-force attacks and plaintext-ciphertext comparison attacks more difficult to accomplish

Here is how DES-X works:

- A message block is XOR-ed with the first 64-bit key
- The resultant data block is DES-encrypted using a 56-bit key
- The resultant ciphertext is XOR-ed with the third (again 64-bit) key

# Des-X

In theory DES-X lengthens the key to 184 bits (64+56+64)

In practice however you need to subtract a logarithm of the number of selected plaintext pairs and corresponding ciphertexts, which can lower the actual key strength to 88 bits

Because DES-X is more secure than the original algorithm, and is not really significantly slower, it was adopted in EFS encryption in older versions of Windows

# 3des

3DES, as the name suggests, is the DES algorithm applied three times to the same data block:

- A data block is first encrypted using the first key
- Next, it is decrypted using the second key
- And encrypted again, using the third key

Because encryption and decryption are symmetric operations, the second phase does not enhance the algorithm or make it stronger

The basic flaw of 3DES is its low efficiency: it is three times slower than DES and only about two times stronger than this algorithm

The practical security of 3DES is about 100 bits of strength (the factual strength of 3DES has not been verified yet): **it is the only DES algorithm than can still be used now**

RC ciphers have been created by Ron Rivest

**RC2** was developed in 1987 as a commission from Lotus

The mechanism behind the algorithm remained secret until 1996, when it was anonymously published on Usenet

The flaws include using 64-bit blocks and vulnerability to chosen-plaintext attacks

The actual strength is 34 bits, which makes it not recommended

**RC5** was developed in 1994 for RSA

You can alter many variables, including round number as well as block size and key length: round number ranges from 1 to 255, the size of a single block can equal 32, 64 or 128 bits, while key length may equal from 0 to 2,040 bits

## How it works:

- A key is split into subkeys $K_0, K_1, . . . , K_{2r}, K_{2r+1}$, where r is the selected round number
- The message is split into two halves $L_0 + S_0$ and $R_0 + S_1$
- The first two subkeys are combined with both parts of the message
- The following operations are executed in each round:

$$L_i = ((L_{i-1} \text{ XOR } R_{i-1}) <<< R_{i-1}) + K_{2i}$$
$$R_i = (R_{i-1} \text{ XOR } L_i) <<< L_i) + K_{2i+1}$$

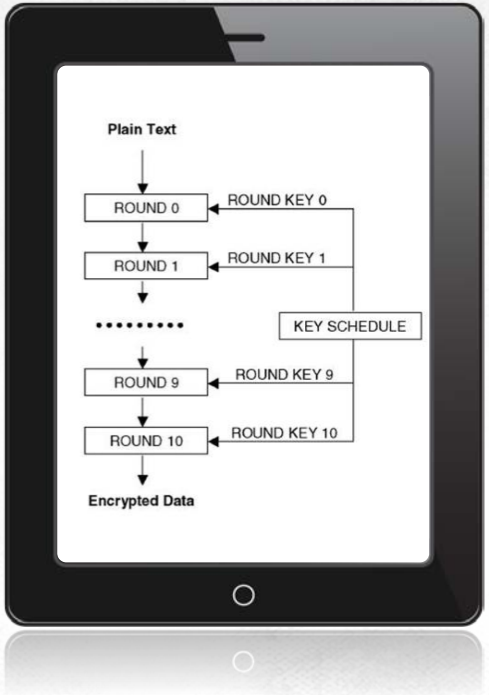
**Both RC5 and its modification,** RC6, were entered in a NIST competition that picked a DES successor and are not considered too secure

# Advanced encryption standard

Designed by two Belgians (Vincent Rijmen and Joan Daemen) as an entry in the NIST competition for a new cryptographic standard

The goal of this 1997-2000 competition was to find a replacement for DES, which had been in use for thirty years at that time

Shortlisted, Rijndael trumped MARS, RC6, Serpent and TwoFish

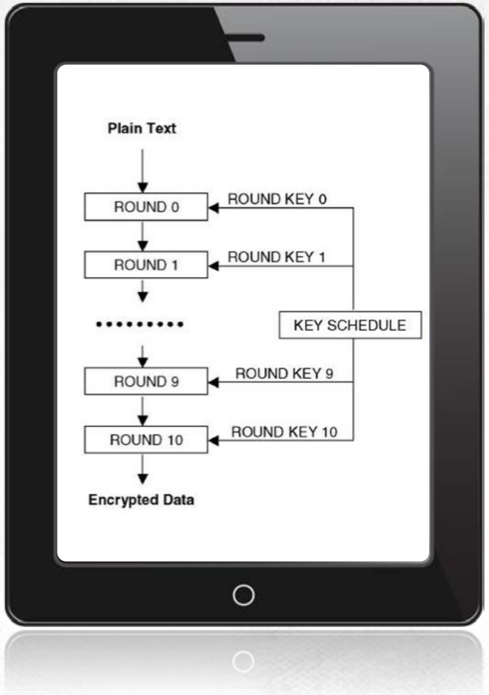


# Advanced encryption standard

Round number depends on key length: a 128-bit key takes 10 rounds, a 192-bit key takes 12 rounds, while a 256-bit key takes 14 rounds

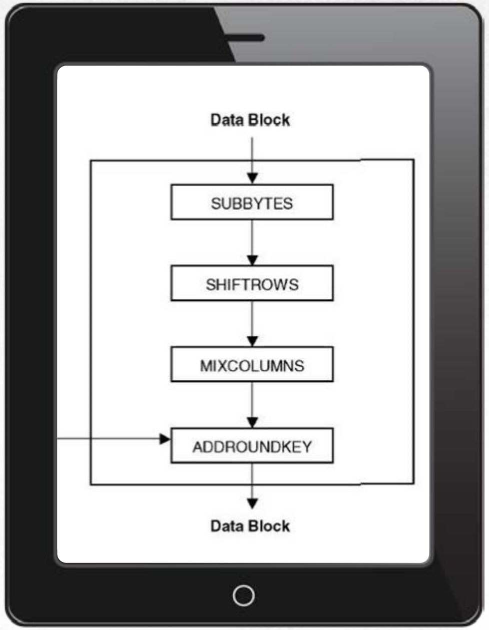
The transformations made in the rounds do not follow Feistel's network

Each round (except the last one) is made up from four different reversible, homogeneous transformations that create three layers: the non-linear layer, the mixing layer (where two transformations occur except for the last round with only one linear transformation) and the key addition layer



# Advanced encryption standard

The changing plaintext values at the end of each algorithm transformations are referred to as states. States can be represented as tables where cells are message bytes. Also a key is represented as a table



# Advanced encryption standard

Encryption in each round:

- The non-linear SubByte transformation occurs. It is applied independently to each state byte and is a 8 x 8 S-box. Each S-box is a reversible transformation that is the combination of two non-linear operations
- The linear transformation ShiftRow is occurs. It periodically shifts rows in a state table by a specific number of bytes: the zero row is not shifted, the first row is shifted by 1 byte, the second by 2 or 3 bytes (depending on block size), and the third by 3 or 4 bytes (depending on block size)
- In all rounds except the last one, the MixColumn linear operation is carried out. In this transformation, state table columns are treated as polynomials and multiplied modulo with a fixed polynomial
- AddRoundKey is the last operation, which involves XOR-ing round key bytes with state bytes

**AES is both a fast and secure cipher and the current standard for symmetric cryptography**

## Block cipher modes

To encrypt a variable-length message using a block cipher, you first need to split it into fixed-length blocks equal to the size of a block transformed by a given algorithm

**Block cipher modes strongly influence the security of a ciphertext**

Here are the most popular modes of transforming data streams into fixed-length blocks:

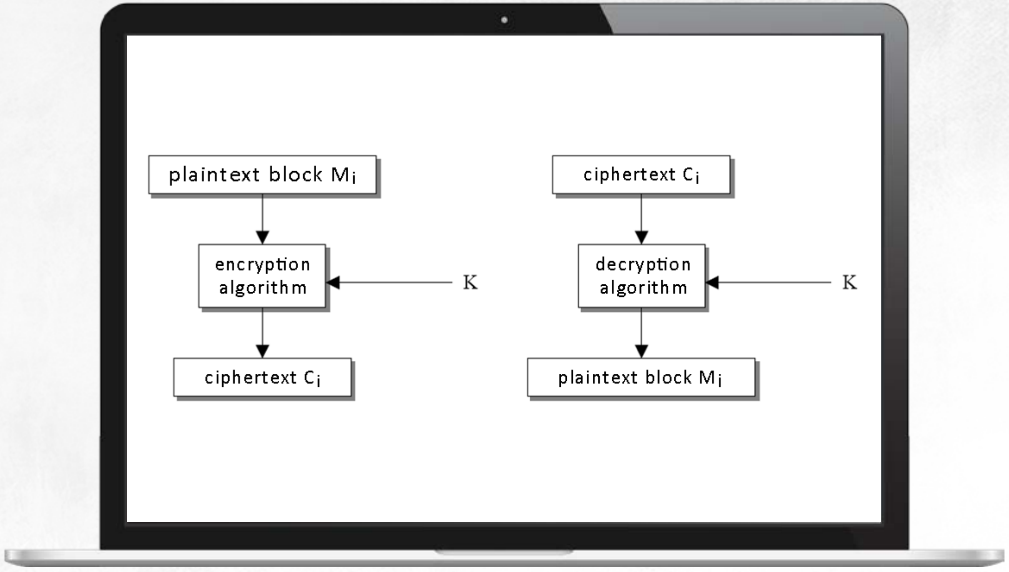
- Electronic codebook mode (ECB)
- Cipher-block chaining (CBC)
- Counter mode (CTR)
- Feedback modes (CFB and OFB)

# Block cipher modes

## Electronic codebook mode: ECB

The most basic block cipher mode

The plaintext is split into fixed-length blocks of bits and the blocks are then separately encrypted using the same key

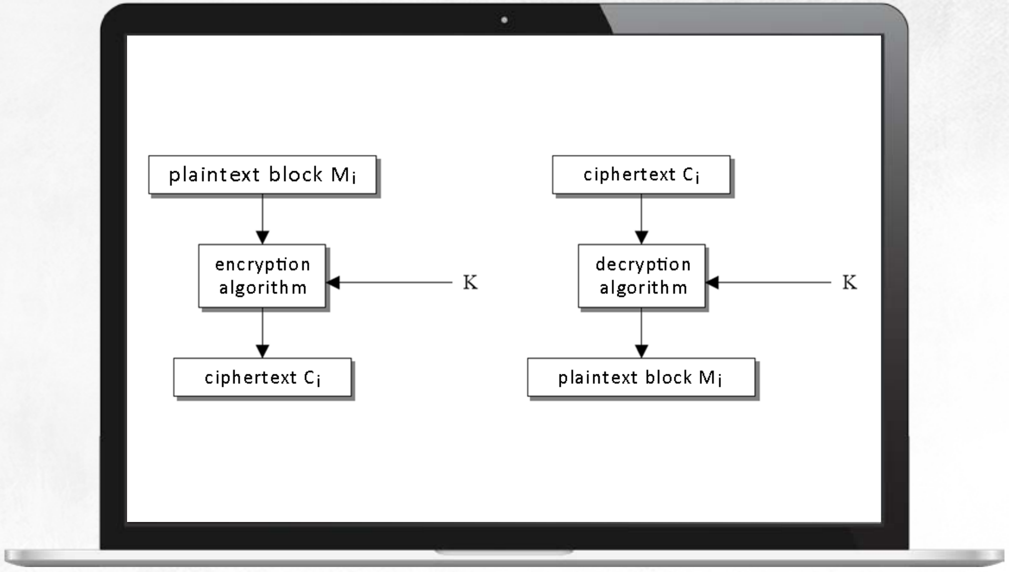


# Block cipher modes

## Electronic codebook mode: ECB

The fundamental flaw of ECB that makes it unusable is that equal plaintext blocks generate equal ciphertext blocks. This discloses a lot of valuable information about the cipher and the original message

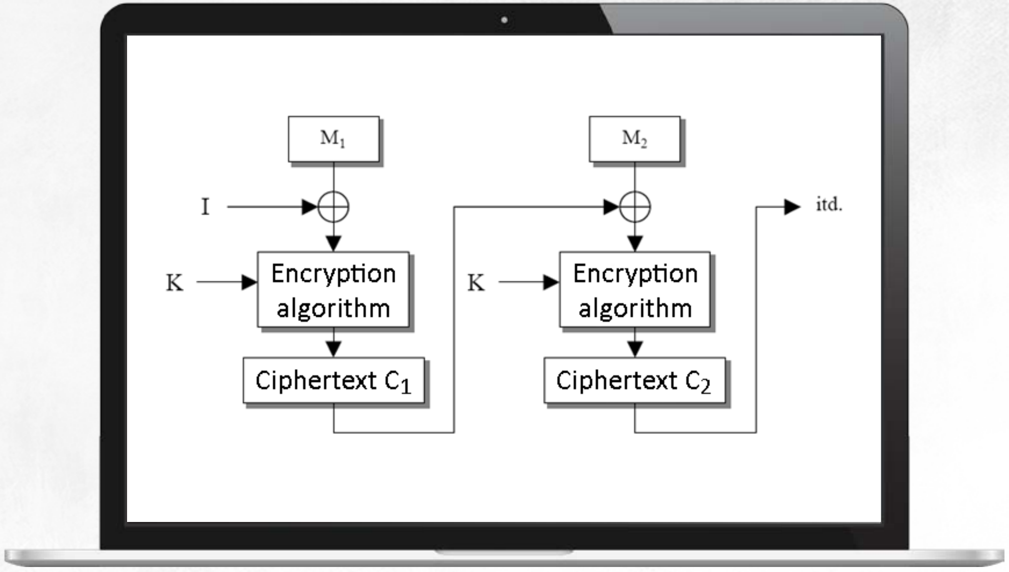
What's more, attackers can transparently modify the ciphertext and add additional blocks to it, for example by repeating the first block multiple times



# Block cipher modes

## Cipher-block chaining: CBC

Each message block, before encryption, is XOR-ed with the ciphertext that is generated from encrypting the previous message block, while the first block is encrypted using an initialisation vector

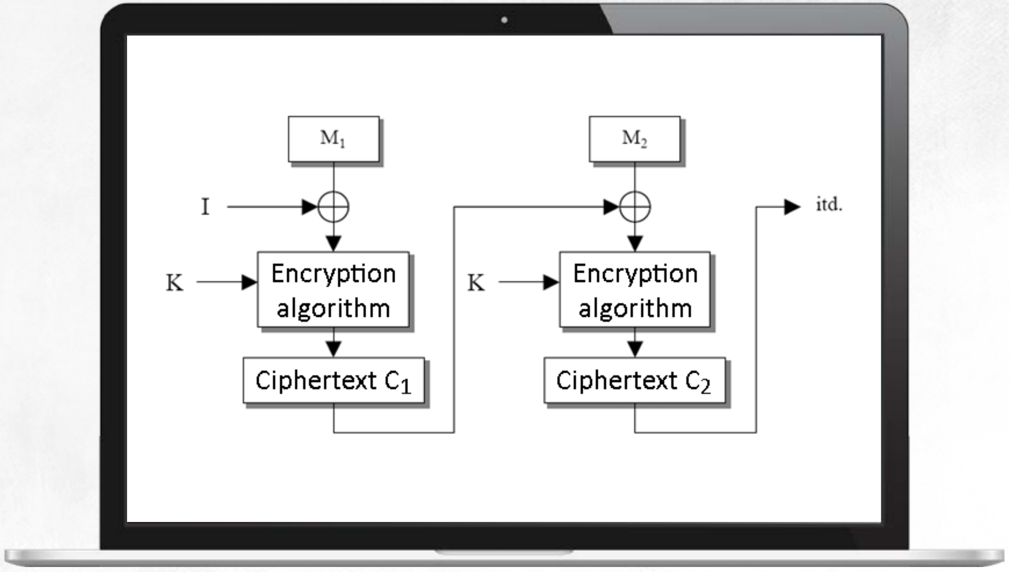


# Block cipher modes

## Cipher-block chaining: CBC

This mode is used by default in Microsoft solutions. Here are the features of CBC:

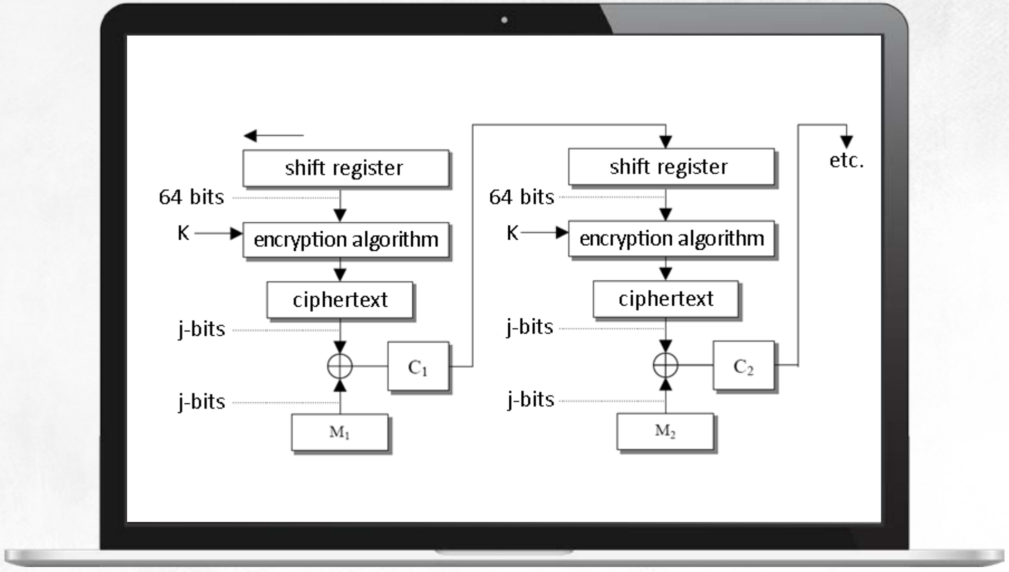
- The IV doesn’t need to be encrypted (its ciphertext would be as random as the IV itself)
- Encrypting the same message using the same key twice will generate two different ciphertexts (because a randomised IV was used)
- Even if one block is corrupted, the remaining ciphertext blocks may be still restored: since the corrupted block was used to encrypt the next block, if it is used to decrypt it, the damage can be eliminated



# Block cipher modes

## CFB and OFB

The feedback modes work in a slightly different way: they change a block cipher into a generator of pseudorandom bits and then use the XOR operation to combine a generated string with plaintext bits

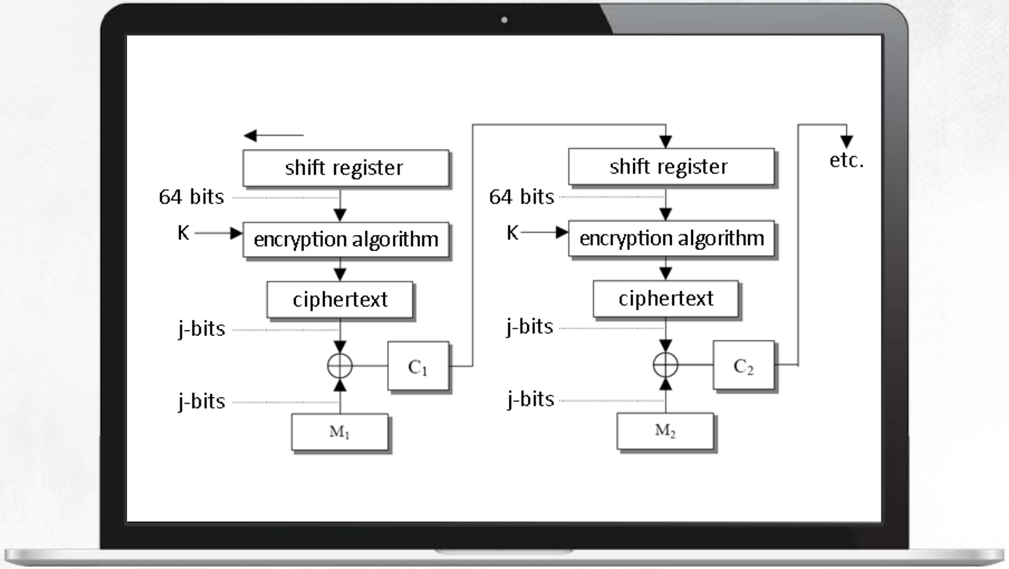


IT SECURITY ACADEMY
www.SecAcademy.com

# Block cipher modes

## CFB and OFB

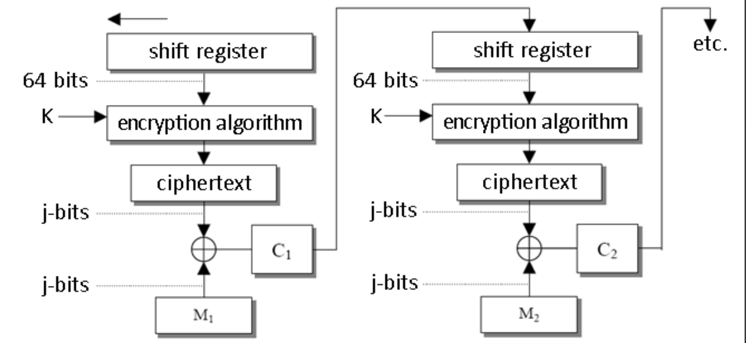
CFB uses a shift register with an initialisation vector. This IV is the input for an encryption function. Next, j-bits at the left side of function result are XOR-ed with the first block of j-bits of plaintext $M_1$, and the output is the first encrypted block $C_1$. $C_1$ is then shifted into the shift register (at the right side), and shifting to the left by j-bits is carried out the register. Encrypting the next j-bit blocks follows the same schema



# Block cipher modes

## CFB and OFB

The difference between CFB and OFB is that in OFB it is the j-bit block from an encryption function result is shifted to the right side of a shift register, and not the j-bit block from an encrypted block



IT SECURITY ACADEMY
www.SecAcademy.com

# Block cipher modes

## CTR

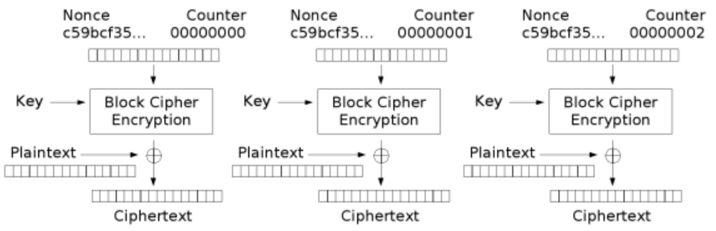
### Popular block cipher mode

Enables decrypting a selected block without needing to decrypt the full ciphertext

### Similar to OFB:

- An IV is chosen
- The IV is encrypted
- The resultant ciphertext is XOR-ed with the plaintext

The cipher generates a key which as long as the plaintext, and which is later used for the XOR operation



Counter (CTR) mode encryption

# Stream ciphers

## RC4

The three last block cipher modes made block ciphers into stream ciphers: **if a shift equals 1 bit, one bit at a time is encrypted and decrypted**

There are also symmetric stream ciphers that encrypt one bit at a time

An example of a cipher of this type is **RC4**. Developed in 1987, the mechanism behind it was published on the Cypherpunks mailing list in 1994

While it is difficult to implement properly, and it is vulnerable to linear and differential attacks, RC4 remains widely used today

# Stream ciphers

## RC4

Here's how RC4 works:

- A pseudorandom bit stream is generated (a keystream)
- At encryption, the keystream is XOR-ed with the plaintext
- Decrypting takes the same XOR operation

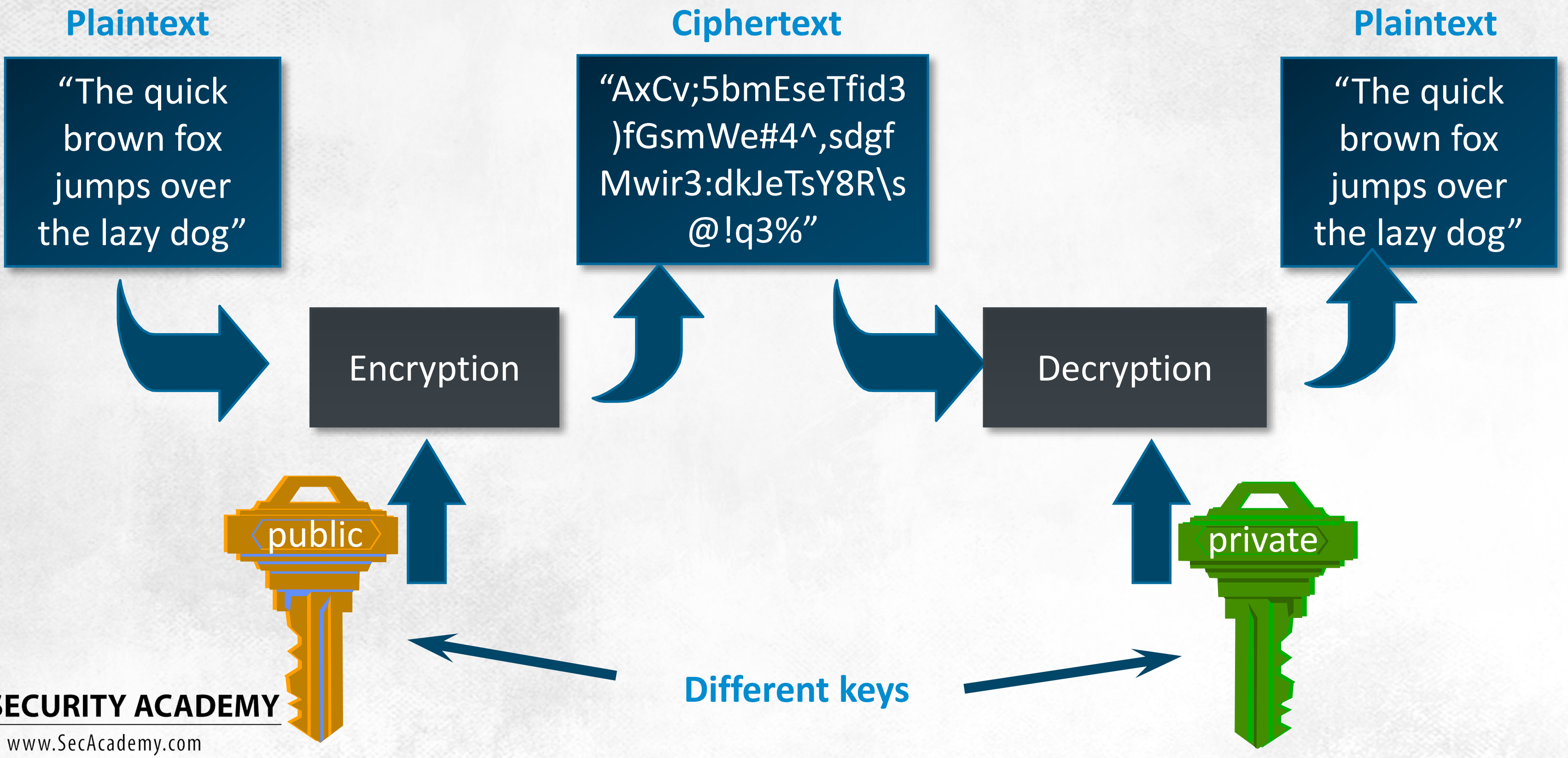
A secret internal state is used to generate the keystream. It is comprised of two parts:

- A permutation of all 256 possible states
- Two 8-bit index-pointers

The initial permutation makes use of a key that has a size of from 40 and 256 bits. A keystream is created based on this key. The generation involves modifying the state table as many times as many there are bits in the plaintext

**If the same message** is encoded twice using the same key, you'll get equal ciphertexts

# Asymmetric-key algorithms


Plaintext
"The quick brown fox jumps over the lazy dog"
Encryption
public
Ciphertext
"AxCv;5bmEseTfid3)fGsmWe#4^,sdgfMwir3:dkJeTsY8R\s@!q3%"
Decryption
private
Plaintext
"The quick brown fox jumps over the lazy dog"
Different keys

## Asymmetric-key algorithms

Pros:

- Make secure key exchange possible
  - Each user only needs two keys: the public key (known by all system users) and the private key (known only by its owner)
  - You cannot recover the private key easily based on the public key
- Reduce the number of keys
- Enable trust building

Cons:

- Low efficiency
  - Have not been build with encrypting long messages (like files) in mind and because of this, if misused, they drastically reduce the security of ciphertexts
- Require much longer keys than symmetric keys
  - The strength of a 512-bit asymmetric key is more or less equivalent to the strength of a 60-bit symmetric key

# Asymmetric-key algorithms

Extending trust relationships from computers to people (organisations) is problematic
Characteristic feature:

- Modulo (instead of XOR)

Asymmetric ciphers rely on one-way functions, operations that are easy and efficient to compute in one direction but cannot be easily inverted

The ElGamal and DSA ciphers rely on discrete logarithm complexity (the a-base discrete logarithm of the element b, in a given finite group is an integer c such that $a^c=b$). If the modulo operation only requires performing O(log c) operations, the only way to compute a discrete logarithm is to check all possible c integers.

The RSA algorithm's security is relies on the fact that multiplying is easy, while the breakdown of a product into factors, factorisation, has high computational complexity

# RSA

Developed in 1977 by Rona Rivest, Adi Shamir and Leonard Adleman

RSA Data Security Inc. was granted a patent for it that expired September 20, 2000

The mechanisms for generating a key pair (a private and public key):

- Pick two large primes p and q. Each prime should have at least 1,024 bits, but just for this example we'll pick these values: p = 7, q = 11
- Compute n = p*q. For our example, it's n = 77
- Choose an exponent e such that (GCD(e, (p–1)(q–1)) = 1; where GCD is the greatest common devisor. In practice, you usually choose the smallest of possible e values: choosing GCD doesn't affect the security of the ciphertext but may make quicken the encryption. For our example, it's **e = 37** (GCD(37, (7–1)(11–1)) = 1)
- Values n and e are the public key. Our public key are numbers 77 and 37
- Choose a number d such that e*d = 1 (mod (p–1)(q–1)). The number d should not be too small. In our case, it's **d =13** (37*13 = 481 = 1(mod 60)).
- The number d is the private key. **Our private key is 13**

# RSA

The factorisation problem is connected to the number n: determining p and q based on the number n only is computationally complex. As it turns out, you can compute the key d if you have the product of $(p - 1)(q - 1)$.

Since p and q are unknown, an attacker can only deduce them based on the public number n

RSA encryption is a process that treats a message as a number slightly smaller than n and performing the following operation: $s = m^e \pmod n$

Let's assume our message is the number 2 ($m = 2$), which means its ciphertext will be 51 ($s = 2^{37} \bmod 77 = 51$)

Decryption is the $m = c^d \pmod n$ operation

Having the private key (the number d), the recipient of our message will perform this operation: $m = 51^{13} \bmod 77$ and retrieve the original plaintext message 2 ($51^{13} \bmod 77 = 2$)

## ElGamal

Designed in the 80s by the Egyptian cryptographer Taher Elgamal

The ElGamal key generation mechanism:

- Choose a large prime number p
- Choose a random number g such that g< $p^2$. Unlike RSA, p and g may be shared by many users
- Choose a random number x smaller than p
- Compute $y = g^x \bmod p$
- The public key is p, g and y
- The private key is x

# ElGamal

Encrypting the message M:

- Choose a random co-prime integer k from a range up to p-1
- Compute r = $g^k$ mod p and s = $y^k M$ mod p. The ciphertext is the pair (r, s)

Decrypting the ciphertext (r,s) involves computing this:

$$\frac{s}{r^x} = \frac{y^k M}{(g^k)^x} = \frac{(g^x)^k M}{(g^x)^k} = M$$

The security of ElGamal rests on the complexity of computing discrete logarithms

Knowing g, p and y, you cannot compute the exponent to which g was raised, which is necessary to determine x

## Hash Functions

Encryption ensures data is confidential, but cannot ensure its authenticity

Likewise, encryption cannot let you verify the authenticity of a sender, which in turn means that a message recipient cannot prove that it was really the sender who forwarded a message

Both problems (authenticity and non-repudiation) can be solved through using hash functions and digital signatures

# Hash Functions

A hash function is a one-way function that returns message hashes of a strict, specific length

Cryptographic hash functions should make output as random as possible, minimise the risk of returning the same hash for two different messages and make output as different as possible for small changes made to a message

In view of this:

- Computing a hash is easy, but mapping a message to an obtained hash is difficult
- Finding two different messages with the same hash is computationally complex

# Hash Functions

## Birthday attack

The most critical problem hash functions have is that they are vulnerable to a type of brute-force attack called the birthday attack

The name is derived from the well-known paradox: in a group of 23 people the likelihood that two of them share the same birthday (of any year) is P2(365,23) ≈ 0,507, so it is more than 50%

What's more, the likelihood of a match (collision) rises dramatically with an increase in group membership; for a group of 30, it is P2(365,30) ≈ 0,706

# Hash Functions

## Birthday attack

Let's try to calculate this:

- There are 365 days in a year, which means the first collision (birthday matching) will occur roughly after $\sqrt{365}$, or 19 attempts
- In a group of twenty people, you can create n(n–1)/2 (190) pairs, for each of them the likelihood that two people in a pair share the same birthday is 1/365
- The total collision risk is 20(20–1)/2∗1/365, which equals 380/730 (more than 50%)
- The attacks of this type exploit the fact that finding a data set for which the computed hash will be the same as for the original set is multiple times easier than determining the original set
- You can protect against this attack by doubling the length of the hash
- What reduces the popularity of hash functions used to authenticate data transmitted across computers is the necessity of exchanging  hashes computed before and after sending a message between the computers through a secure channel

# Hash Functions

## MD

- MD hash functions were developed by Ron Rivest:
- MD2 was created in 1989, MD4 in 1990, MD5 in 1991
- Each of them returns a hash of 128 bits
- In 2004 attack techniques for exploiting all three versions of MDs were published

**The MD functions are now not considered safe and should not be used**

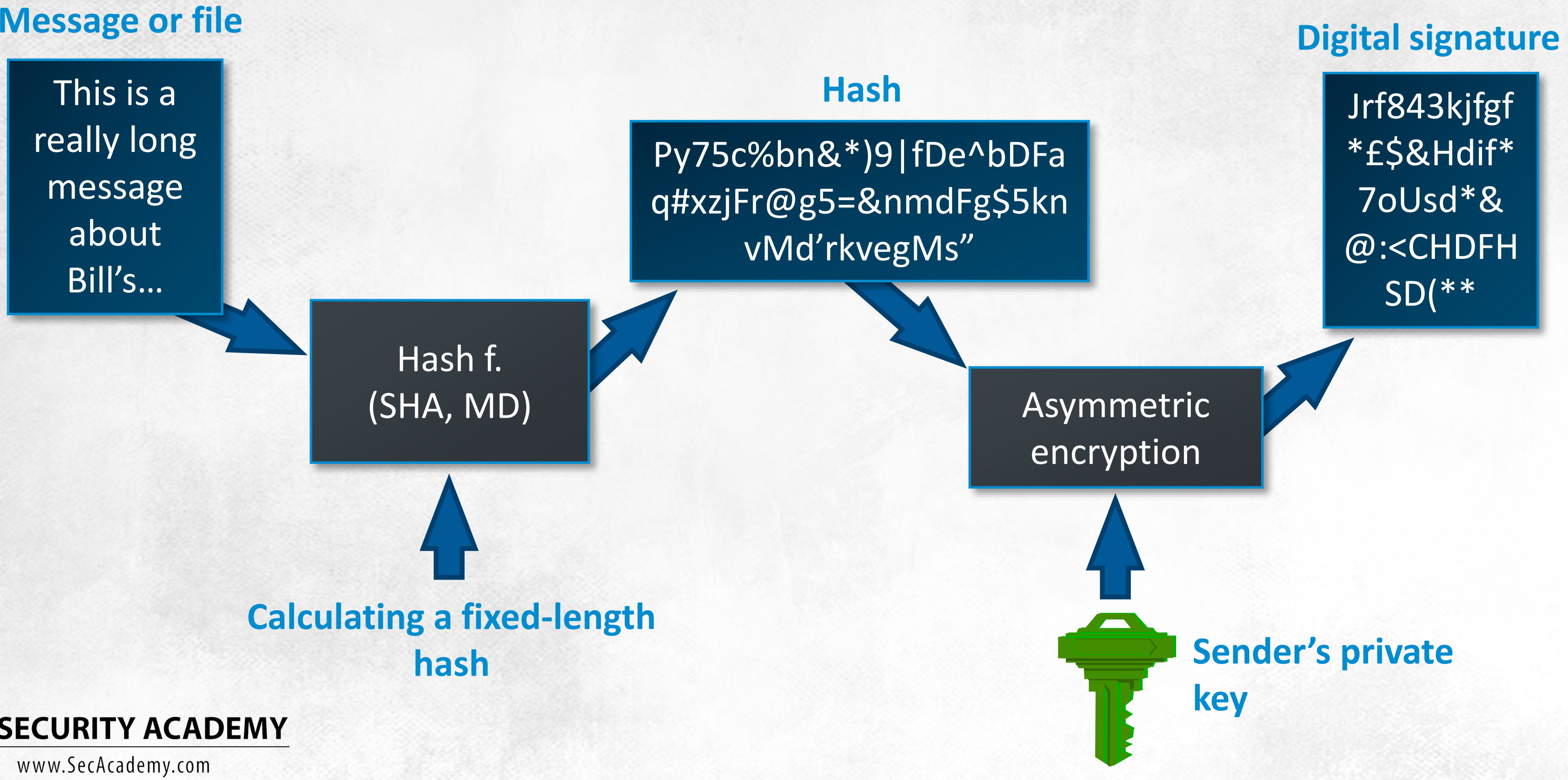
## SHA

- Developed by the NSA, the SHA function returns a message digest of 160 bits
- The two first versions of SHA, SHA-0 and SHA-1, are no longer considered safe to use now: after a new attack technique was published in 2005 SHA-0 only offers a security that is 39-bit strong. The factual security of SHA-1 is 63 bits
- Due to these deficiencies, works on the third version of SHA (SHA-2) were accelerated. SHA-2 generates 256-bit message digests (SHA-256) and 512-bit message digests (SHA-512) as well as makes it possible to use AES keys

**The SHA-2 functions are considered safe and are still widely deployed in computer systems**

# DIGITAL SIGNATURE

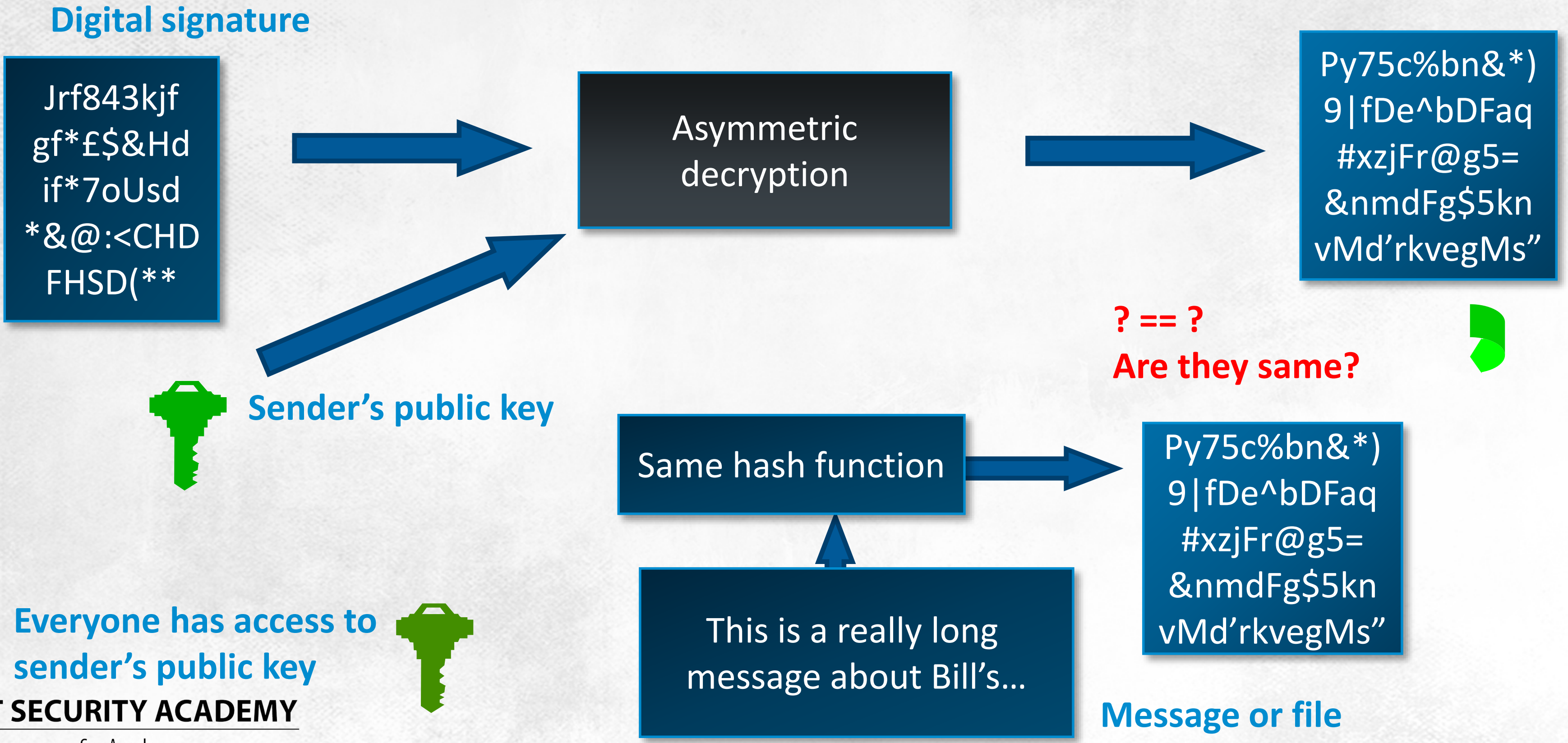
## Signing a message



Message or file
This is a really long message about Bill's…
Hash f. (SHA, MD)
Calculating a fixed-length hash
Hash
Py75c%bn&*)9|fDe^bDFa q#xzjFr@g5=&nmdFg$5kn vMd'rkvegMs"
Asymmetric encryption
Sender's private key
Digital signature
Jrf843kjfgf *£$&Hdif* 7oUsd*& @:<CHDFH SD(**

# DIGITAL SIGNATURE

## Verifying signatures



Everyone has access to sender's public key

# DIGITAL SIGNATURE

## RSA

The security of RSA digital signatures rests on the fact that none of the numbers d (private keys) and numbers e (parts of public keys) are emphasised and may be used both for encryption and decryption

Digital signatures may be generated through the equation $S = (\text{SHA-2}(M))^d \bmod n$, where first the SHA-2 hash of the original message is calculated and then its digital signature is calculated using a private key

Checking the RSA signature is a process that involves calculating its SHA-2 hash and decrypting it using a public key: $\text{SHA-1}(M') = S^e$

# DIGITAL SIGNATURE

## RSA

If you use your private key to encrypt a crafted message (it’ll look like a random string), an attacker will be able to take it and use it to digitally sign any message using your signature. This is how the attack goes:

- The attacker composes a message M that contains info that is advantageous to the attacker (for example a profitable contract)
- The attacker calculates the hash of this message
- The attacker generates a random message X
- The attacker generates a message out of $z*x^e$ where e is the target’s private key, and sends it requesting encryption
- The target decrypts this message using his private key, executing this operation: $(z \cdot x^e)^d$
- Encrypting the message is done by executing the following equation: $(z \cdot x^e)^d = z^d \cdot x^{ed} = z^d \cdot x$. The attacker only needs to divide the obtained ciphertext by x to get the target’s digital signature on the message M

# DIGITAL SIGNATURE

## ElGamal

The private and the public keys cannot be used interchangeably in ElGamal; however, they can be used to sign messages

To sign a message digitally, you need to:

1. Choose a random co-prime k from a range up to p – 1 (with a part of the private key)
2. Calculate r = $g^k$ mod p
3. Sign the message digest with a private key x: s = $k^{-1}$ (SHA-2(M) − xr) mod p – 1
4. The pair r,s is the digital signature

Verifying a digital signature requires you to compute:

- $y^r r^s = (g^x)^r (g^k)^{k-1(\text{SHA-2}(M)-xr)} = g^{xr} g^{kk-1(\text{SHA-2}(M)-xr)} = g^{xr+\text{ SHA-2}(M)-xr} = g^h$ (mod p)
- If the result ($g^h$) equals $g^{\text{SHA-1}(M)}$ mod p, the digital signature is correct

## HYBRID SCHEMES

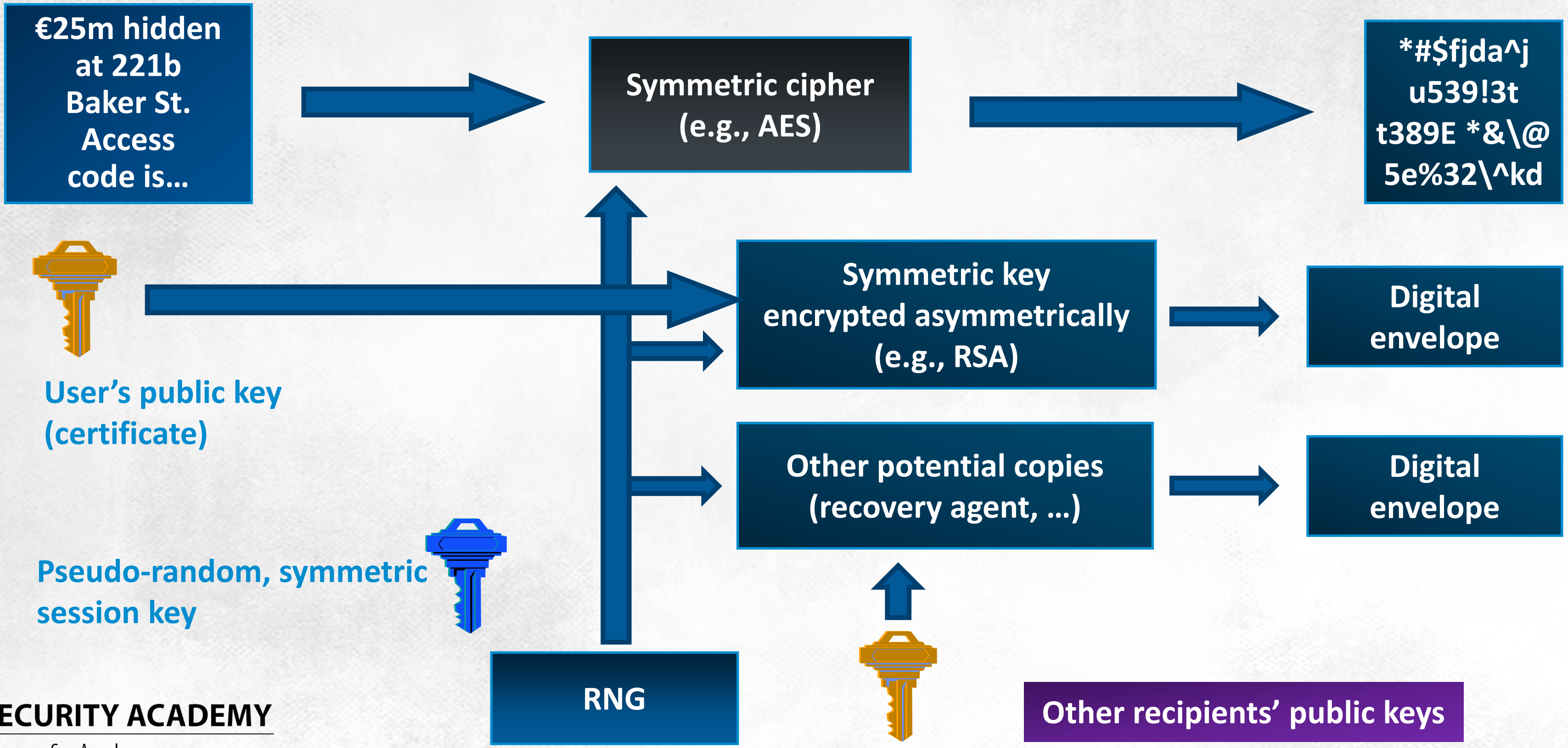
Because symmetric ciphers don’t offer secure key exchange over trusted channels and can’t authenticate messages and identify their signatories but are fast and can be used to encrypt long messages, while asymmetric ciphers cannot be used for long messages but provide a good solution to key exchange problems and for message and signatory authenticity checking, if you combine the best of the two solutions, the resultant  crypto system is fully functional, secure and efficient

Almost all cryptographic systems used nowadays are hybrid in nature, including SSL and EFS

In hybrid schemes messages are encrypted symmetrically, and the keys used for it are encrypted asymmetrically and put into a digital envelope and sent along the ciphertext

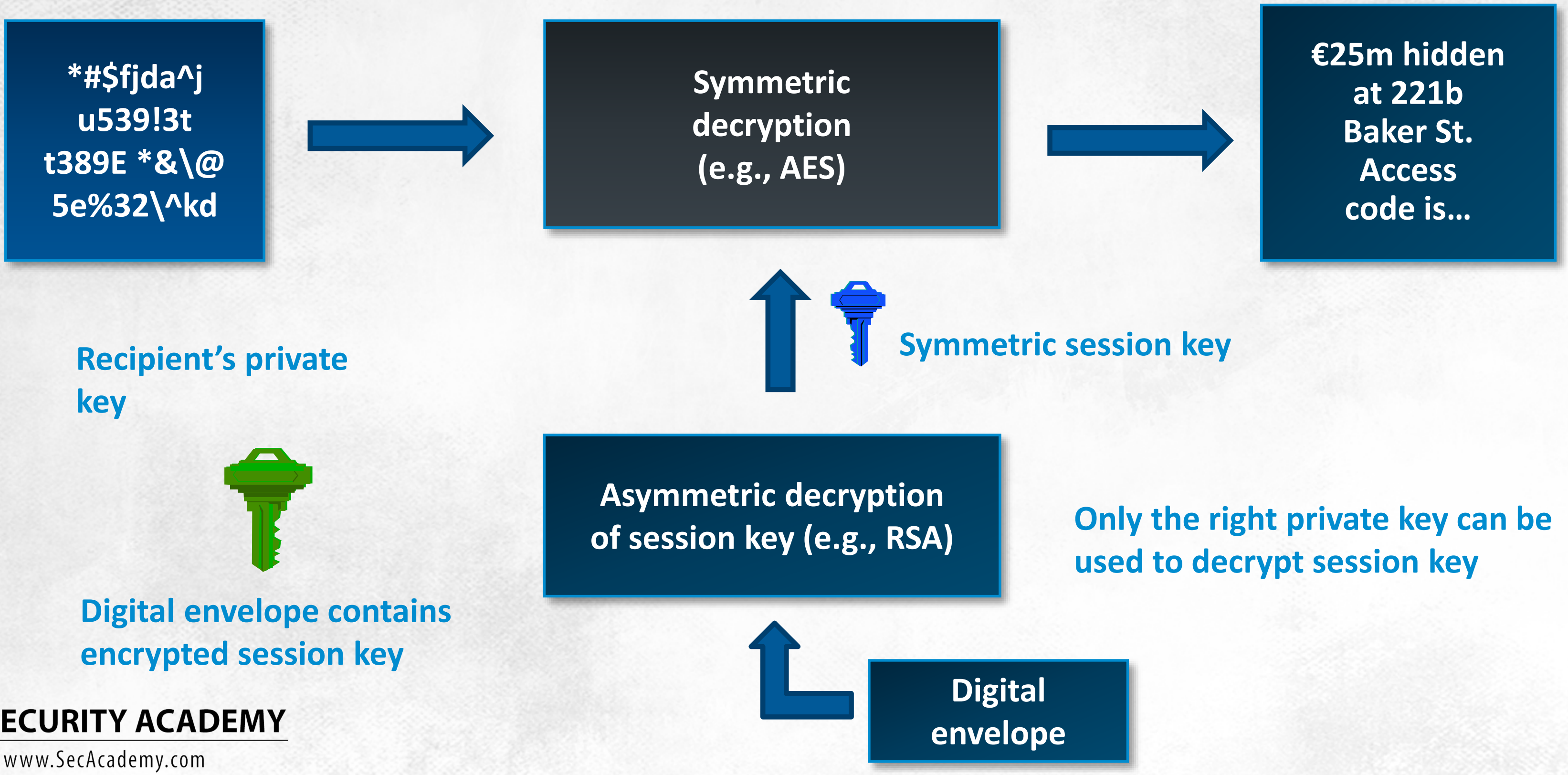
# HYBRID SCHEMES

## Encryption


€25m hidden at 221b Baker St. Access code is…
Symmetric cipher (e.g., AES)
*#$fjda^j u539!3t t389E *&\@ 5e%32\^kd
User's public key (certificate)
Symmetric key encrypted asymmetrically (e.g., RSA)
Digital envelope
Other potential copies (recovery agent, …)
Digital envelope
Pseudo-random, symmetric session key
RNG
Other recipients' public keys

# HYBRID SCHEMES

## Decryption


*#$fjda^j
u539!3t
t389E *&\@
5e%32\^kd
Symmetric decryption (e.g., AES)
€25m hidden at 221b Baker St. Access code is…
Symmetric session key
Recipient's private key
Asymmetric decryption of session key (e.g., RSA)
Only the right private key can be used to decrypt session key
Digital envelope contains encrypted session key
Digital envelope

# Cryptographic systems

Creating a secure cryptographic system is extremely difficult to accomplish:

- If you use algorithms that are vulnerable to attacks, the security of the system drops significantly. This means that even if you're a cryptographer/mathematician, **you should never use algorithms you created in live computer systems**. Developing a secure cipher takes a lot of time, and proving it is resistant to attacks will take years. Because of this, **do not use any secret cipher** (whether developed by you or an application's producer): security through obscurity gives you very little protection!
- The practical security of a system is a matter of both using a good algorithm and implementing it in the right way: even the most secure algorithm, if ill-implemented, will not be immune to attacks. This is especially applicable to block cipher modes and confidential data protection methods (above all, keys). Even if you're a programming guru, **do not implement cryptographic algorithms by yourself**: there is a huge gap between a program that 'works' (encrypts and decrypts data) and a program that is actually able to protect confidential information.
- **A cryptographic system is only as secure as its weakest element**: even if a web application is protected with a 4-kilobyte RSA key, it's meaningless if the data sent across a web server and browser is DES-encrypted

THANKS

IT SECURITY ACADEMY
www.SecAcademy.com