# What is (modern) cryptography?

The term **cryptology** refers to the science that studies secret writing, that is, the errors that, when processed in a certain way, become difficult or impossible to reach by unauthorized entities. It is made up of two disciplines: **cryptography** (which deals with the study of algorithms, protocols and systems that are used to protect information and provide security for communications and entities that communicate) and **cryptoanalysis** (which is concerned with capturing *cryptograms* -that is, the "protected" texts built by cryptography- without having authorization and, then, trying to obtain the corresponding *clear text* by using specialized techniques).

Cryptography appeared after World War II is the one considered "modern", as this was the moment when specialized machines (that is, computers) began to be used to generate cryptograms and/or to break them in an automatic way. These machines forced to completely change all known methods used for creating cryptograms because these became obsolete and vulnerable.

## Problems solved by cryptography

For a communication to be considered secure, the chosen cryptographic method must guarantee that this communication has the following characteristics:

- **Confidentiality**: By *encrypting* a message "A", a different message "B" is obtained as a result, which should have a meaning only known by who can *decrypt* it (obtaining the original message "A", then). The message "B" can be stored or be transmitted to a specific destination: the idea is that a receiver, having obtained the message "B", cannot decipher it (that is, understand it) if he/she is not the legitimate receiver
- **Authentication**: The objective is to ensure that the sender of the message is who really said it was
- **Integrity**: The objective is to detect a possible alteration of the original message
- **No repudiation**: The objective is to prevent a sender from denying being the author of a message sent by him previously (only some algorithms of asymmetric type offer this feature)

## Types of cryptographic algorithms

- **Symmetrical:** It uses the same (binary) key for both the encryption algorithm and the decryption algorithm; that's why this key is also called *"shared"* key (between the sender and the receiver of the message). This kind of cryptography is fast and secure but its biggest problem is how to distribute that shared key between the ends via an insecure channel before secure comunication could begin; to solve this it's necessary the use of asymmetrical protocols (read next pharagraph).

- **Asymmetrical:** It uses two keys: one (*public*) in the encryption algorithm and another (*private* but linked to the previous one) in the decryption algorithm. This allows not to have to share any key between the two extremes because each one uses a different one. Both public and private keys are usually implemented in the form of binary files.

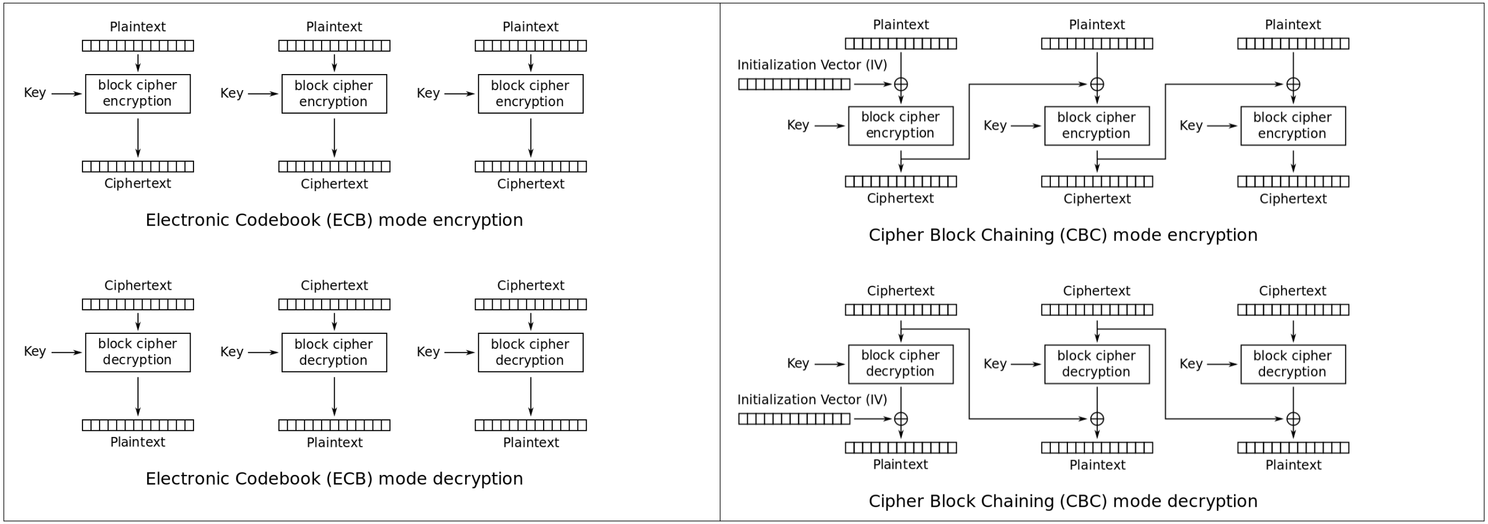
## 1.Confidentiality in symmetric cryptography

To encrypt/decrypt, various symmetrical algorithms are available to choose from, each of which can be generically classified into one of the following major categories:

- **Stream:** They encrypt bit by bit. They are useful in real-time data flows with little structure. Examples of algorithms: **XOR**, **RC4**, **ChaCha20/12/8**, **Salsa20/12/8** ...
- **Block:** Instead of encrypting a bit each time, they create a block of bits (whose size can be 64 bits, 128 bits...depending on the specific chosen algorithm) and then encrypt the block as a whole. Examples of algorithms: **AES** (with various modes of operation -see next slide- and various standard key sizes: 128 bits, 256 bits... -in general, the longer the better-), **Blowfish**, **Twofish**, **Serpent**, **CAST**, **LOKI**, **DES**, **3DES** ...

## Modes of operation in a block algo.

What happens if you have to encrypt more than one block? You must choose an "operating mode", which determines how to connect the blocks. This fact is important because depending on the chosen mode of operation, the security of the message can be compromised although the chosen block algorithm is optimal. Examples of modes are: **ECB** (very insecure due to existence of repetitive patterns in cipherblocks), **CBC** (which solve this problem of the patterns' existence by mangling blocks between them), **OCB**, **CFB**, **OFB**, **CCM**, **CTR** or **GCM**.


Plaintext
Key
block cipher encryption
Ciphertext
Electronic Codebook (ECB) mode encryption
Ciphertext
block cipher decryption
Plaintext
Electronic Codebook (ECB) mode decryption
Initialization Vector (IV)
Cipher Block Chaining (CBC) mode encryption
Cipher Block Chaining (CBC) mode decryption

*The IV is a random number used to obtain a different cipherblock each time the first clearblock is read although this was always the same*

## AE/AEAD block algorithms

The GCM mode belongs to the category of **AE** algorithms (from *"Authenticated Encryption"*), which are designed to provide not just confidentiality but also data integrity and authentication. That is: an AE encrypt algorithm provides integrity and authentication of messages in the symmetrical algorithm itself, while non-AE encrypt algorithm must rely on external algorithms (in particular, hashes signatures) to guarantee that.

**AEAD** (from *"Authenticated Encryption with Associated Data"*) is a variant of AE that allows checking the integrity+authentication of encrypted *and not encrypted* information in a message. This is required for securing network packages: their payload needs confidentiality and integrity while their header just needs integrity because it must be visible for routers. In fact, the TLS 1.3 protocol only allows AEAD algorithms, which means that *AES-GCM/AES-CCM* and *ChaCha20-Poly1305* are the only options available.

## 2.Authentication in non-AE/AEAD

By the very definition of a shared key, only the ends that know it can establish secure communication between them. Therefore, authentication is implicit.

Another thing is that this key could be stolen by some intruder... but this would be the same situation that if a private key is stolne in asymmetric cryptography paradigm: the entire security structure will be compromised.

## 3.Integrity in non-AE/AEAD

The most common mechanism (if the algorithm used is not AE/AEAD) to detect that an original (and already encrypted) message has not been manipulated is:

1. Sender has to calculate the **"hash"** of its content
2. Sender has to send the original message with the "hash" attached to it.
3. Receiver has to recalculate the "hash" of the received message and compares to the received attached "hash": if they are equal, this means that the original message has not been modified; if not, the message will be refused

A "hash" is a string of characters, obtained by applying over any arbitrary amount of data a specific algorithm (like **MD5**, **SHA1** or **SHA2**) that has to achieve this:

- Regardless of the length of the input data, the output hash value must always have the same fixed length (which depends on the specific algorithm used)
- Any change to the original data , no matter how minor is, has to cause a new, completely different output (that is, a brand new string of characters). This fact makes hashes very useful to easily check the integrity of any data.

## 3BIS.HMACs

However, how can we guarantee that the hash itself hasn't been manipulated (or substituted) after having maliciously modified the original message? The solution is to **encrypt the hash too, with a symmetrical key** (which can be the same than the one used to encrypt the message, or another one): this will guarantee that this "hash" has actually been created by the sender, since it is the only one that connects the key.

The functions that allow generating these "encrypted hashes" (also connected with **HMACs**, from *Hash-based Message Authentication Code*), unlike the "classic" "hash" functions, do not have just a single parameter (the original message to "hash") but rather they have two (the original message and the symmetrical key used to encrypt the generated "hash"). Most HMAC algorithms are adaptations of the "classic" ones: **SHA1**, **SHA2**, **SHA3** and the family **"Keccak"**, **Poly1305**, **BLAKE**, **RIPEMD**,...

## Summary of the symm. process

In summary, the complete process that allows confidentiality, authentication and integrity in a communication by using symmetrical cryptography would be:

1. Sender encrypts the message to be sent with a shared key (we'll name it "key1"). If using an AE/AEAD algorithm (such AES-GCM), this step also ensures the authentication and the integrity of the message
2. If an AE/AEAD algorithm is not used in former step, then sender has to calculate the "hash" of the message and encrypt it using another shared key (preferably different from "key1"; we'll name it "key2"). The result of this is called "HMAC"
3. Sender transmits the encrypted message and, if applicable, the HMAC too
4. If an AE/AEAD algorithm was not used in first step, then receiver has to decrypt the HMAC with "key2" to obtain the received hash in clear form. Then, he/she has to recalculate the hash of the received message (still encrypted) to compare both hashes. If the comparison is successful, the integrity of the message is guaranteed and, therefore, the message is accepted (so process can continue in following step).
5. Receiver decrypts the message using "key1"

## 1.Confidentiality in asymmetric cryptography

The basic idea of asymmetric cryptography is that data encrypted with a public key (belonging to the receiver) can only be decrypted with the private key of that receiver. It is to say:

1. Receiver first sends his/her public key (which can be known by anyone) to the eventual senders
2. Sender uses this public key to encrypt the message
3. Sender sends the encrypted message to the receiver
4. Receiver uses its own private key (only known by himeself/herself and no one else) to decrypt the message

As it may be seen, in this procedure private keys are never transmitted and the exchange of public keys is simple because it's not necessary any secure channel for their distribution

The main drawback of public key cryptography compared to symmetrical cryptography is the consumption of resources: encryption/decryption of the former is much more expensive in CPU time.

The asymmetric algorithm currently most used for encrypt/decrypt is **RSA**.

## 2.Authentication in asymmetric cryptography

Thanks to digital signatures, both the sender and the receiver of a message can verify the authenticity of the other entity. Digital signatures are a concept belonging to asymmetric cryptography: the sender uses his private key to sign the message and the receiver verifies this signature with the sender's public key, previously obtained. It is to say:

1. Sender signs the message to send with his/her own private key (which it not known by anyone else)
2. Sender can also encrypt the message to be sent with receiver's public key
3. Sender sends the message to the receiver
4. Receiver uses the sender's public key to verify the signature and, therefore, ensure that it is he/she, in fact, who has sent it.
5. If the message is encrypted, receiver should use his/her own private key to decrypt it

There are various asymmetric algorithms specialized in the creation/verification of signatures: **RSA** (again), **DSA**, **ElGamal**, **DHE** and the elliptical curve variants of the previous ones (**ECDSA**, **ECDHE**...)

## 3.Integrity in asymmetric cryptography

As with symmetric cryptography, asymmetric cryptography also uses "hashes" to detect that a message has not been tampered during a transmission. The procedure is very similar: the "hash" of the (normally, already encrypted) message is calculated and then this encrypted message is sent with its corresponding "hash" attached to it. When the receiver gets the set "message+hash", he/she will recalculate the hash from the received message and it will compare with the received hash: if they are equal, it will proceed to decrypt the message because it has not been modified; if not, the message will refused.

How can we, however, guarantee that the hash itself is not further manipulated (or substituted by someone else after having maliciously modified the original message)? The solution in this case is to **sign the hash with the sender's private key**, since this will guarantee that this "hash" has actually been created by the sender (the only one who owns that key). So, if the "hash" of a message is signed, the receiver will have to verify this signature (remember, with the public key of the sender) and this implies that if the signed "hash" has been manipulated, receiver could not verify it because the signature would not fit (and message would be refused).

## Summary of the asymmetric process

The complete process that allows confidentiality, authentication and integrity in a communication by using asymmetrical cryptography would be:

1. Sender calculates the hash of the message to be sent
2. Sender signs the hash with his/her private key and attaches this signed hash to the message
3. Sender encrypts the message with the receiver's public key (normally including the signed hash inside).
4. Sender transmits encrypted and signed message.
5. Receiver decrypts the message by using his/her own private key.
6. Receiver checks the hash signature by using the sender's public key.
7. Receiver recalculates the hash from the already decrypted received message and compares it with the already verified, received hash
8. If the comparison is correct, the message is accepted

## Hybrid cryptography

Hybrid cryptography combines the shared key system and the public key system to obtain all its advantages without its disadvantages. In this system, public key cryptography (which is computationally demanding, remember) is used just to establish a secure channel to transmit then only a short shared key to the receiver, nothing else. This **shared key will normally be generated in a dynamic and ephemeral way** just for this communication (therefore, an intruder who could guess the shared key embedded in a given communication will not be able to use it for any other communication, neither past nor future). Once both ends have agreed (thanks to the secure channel established by public key cryptography) to use a specific shared key, symmetrical cryptography (which is more quick and less computationally expensive than asymmetrical one) will be used since then to encrypt the whole conversation.

Among the protocols possible to carry out this prior negotiation for the secure exchange of the shared key in an insecure channel and in an unauthenticated manner, the **"DH"** protocol stands out (or "DHE" from "ephemeral") and their variants of elliptical curve algorithms (**ECDHE**).

This way of working is what, for example, the **SSH** program or the **TLS** technology use