Nocions de criptografia

Què és la criptografia?:

La paraula *criptografia* ve del grec 'kryptos' (amagar) i 'gráphein' (escriure); és a dir, significa 'escriptura amagada'. La criptografia és la ciència matemàtica que estudia els mètodes possibles per amagar el significat d'un determinat missatge (és a dir, per 'xifrar-lo') i així poder-lo enviar/emmagatzemar a través de medis/suports insegurs amb l'objectiu de que només el destinatari autoritzat el pugui entendre (mitjançant un procés de 'desxifratge' que només ell podrà fer).

El *criptoanàlisi* és la ciència matemàtica que estudia els mètodes possibles per 'trencar sistemes criptogràfics' (és a dir, per esbrinar el significat d'un missatge xifrat tot i no ser el destinatari legítim). La *criptologia* és l'estudi científic que engloba tant la criptografia com el criptoanàlisi.

No confondre 'criptografia' amb '*esteganografia*'. Aquesta darrera és la ciència matemàtica que permet amagar la pròpia presència del missatge dins d'un determinat suport.

Problemes que ha de resoldre la criptografia moderna:

Per a què una comunicació es pugui considerar segura, el mètode criptogràfic utilitzat haurà de garantir que aquesta comunicació tingui les següents característiques:

Confidencialitat: En xifrar un missatge "A" (és a dir, en voler ocultar el seu significat) s'obté com a resultat un missatge "B" diferent de l'original que tindrà un significat només conegut per qui pugui desxifrar-lo (tornant a obtenir així el missatge original "A" de nou). El missatge "B" (és a dir, el producte del xifrat) pot ser emmagatzemat o bé transmès a un determinat destí, tan se val: en qualsevol cas la idea és que un receptor que no pugui desxifrar el missatge, encara que el tingui al seu poder, no el pugui entendre.

Autenticitat: L'objectiu és assegurar que el remitent del missatge sigui qui realment diu que és ; això s'aconsegueix, en el cas dels mètodes asimètrics, signant-lo amb una clau privada, única per cada remitent (i que se suposa que no ha sigut robada) i, en el cas dels mètodes simètrics, amb la pròpia clau compartida (que se suposa que tampoc ha sigut robada)

Integritat: L'objectiu és detectar una eventual alteració del missatge original ; això s'aconsegueix amb l'ús de funcions hash

No repudi: L'objectiu és evitar que un remitent pugui negar ser l'autor d'un missatge enviat per ell prèviament.

Tipus d'algoritmes de xifrat-desxifrat:

Per xifrar/desxifrar un missatge s'ha de fer servir un element secret anomenat "clau". Depenent del tipus de clau que es faci servir, es poden classificar els mètodes de xifrat-desxifrat en dos grans tipus:

Simètric: Es fa servir la mateixa clau (binària) pels tant en l'algoritme de xifrat com en el de desxifrat ; per això a vegades aquesta clau s'anomena "compartida" (entre l'emissor i el receptor del missatge).

Asimètric: Es fa servir un parell de claus: una clau (pública) en el algoritme de xifrat i una altra (privada però vinculada a l'anterior) en el de desxifrat. Això permet no haver de compartir cap clau entre els dos extrems perquè cadascun en fa servir una de diferent. Tant les claus públiques com les privades solen implementar-se en forma de sengles fitxers binaris.

De l'elecció del tipus d'algoritme de xifrat-desxifrat se'n derivaran els algoritmes emprats als altres àmbits que afecten a la criptografia més enllà de la confidencialitat (és a dir: autenticitat, integritat i no repudi).

La longitud (en bits) de les claus (l'anomenarem n) determina la quantitat de combinacions/claus diferents (2^n) que es poden generar amb força bruta per intentar desxifrar amb cadascuna d'aquestes claus generades un determinat missatge xifrat. En principi, doncs, quant més llarga sigui la clau, més difícil serà trobar la clau correcta. En algoritmes simètrics la longitud mínima acceptable avui dia és de 128 bits; en algoritmes asimètrics és de 4096 bits (excepte en els de corba el·líptica, també coneguts com "ECC", que és 256 bits). No obstant, cal tenir en compte també que com més llarga sigui la clau, més cost de computació haurà d'assumir la màquina a l'hora de xifrar/desxifrar. De fet, per exemple,

utilitzar una clau RSA més llarga de 2048 bits se sol considerar una "despesa" inútil perquè no aporta més seguretat (per la pròpia natura de l'algoritmo) però sí més cost. Per veure diferents recomanacions de tamanys, es pot consultar la web https://www.keylength.com/en . En aquest sentit, very generally speaking, to get the same security strength as an *N*-key symmetric cipher, an elliptic curve algorithm or a cryptographic hash needs to have a key *2N* times as long while a "classical" asymmetric cipher needs to have a key roughly *10N* times as long or longer. És per això que, en el cas de la criptografia asimètrica, en escollir entre algoritmes "clàssics" basats en la factorització de números com és el RSA i algoritmes "ECC", se sol escollir aquests darrers per tal d'optimitzar temps i energia de computació.

Confidencialitat en criptografia simètrica:

Els algoritmes de xifratge simètric utilitzats en l'actualitat es poden classificar en dos tipus:

De flux (stream) : Xifren bit a bit. Són útils en fluxos de dades en temps real amb poca estructura. Exemples: **XOR** -elemental-, **RC4** -obsolet-, **ChaCha20, Salsa20...** Per saber més mireu https://ctf101.org/cryptography/what-are-stream-ciphers

A stream cipher is an encryption algorithm that encrypts 1 bit or byte of plaintext at a time. It uses an infinite stream of pseudorandom bits as the key. For a stream cipher implementation to remain secure, its pseudorandom generator should be unpredictable and the key should never be reused. Stream ciphers are designed to approximate an idealized cipher, known as the One-Time Pad. The One-Time Pad, which is supposed to employ a purely random key, can potentially achieve "perfect secrecy". That is, it's supposed to be fully immune to brute force attacks. The problem with the one-time pad is that, in order to create such a cipher, its key should be as long or even longer than the plaintext. In other words, if you have 500 MegaByte video file that you would like to encrypt, you would need a key that's at least 4 Gigabits long. Clearly, while Top Secret information or matters of national security may warrant the use of a one-time pad, such a cipher would just be too impractical for day-to-day public use. The key of a stream cipher is no longer as long as the original message. Hence, it can no longer guarantee "perfect secrecy". However, it can still achieve a strong level of security.

De bloc : En comptes de xifrar un bit cada cop, xifren un bloc de bits a la vegada. L'algoritme d'aquest tipus més usat amb diferència és l'**AES** (amb diversos modes d'operació i tamanys de claus), encara que hi ha més (com ara **Blowfish, Twofish, Serpent, CAST, LOKI,** ...). Cal tenir en compte que, a més de per xifrar/desxifrar, els algoritmes de bloc també s'usen per construir algoritmes de flux, generadors de números pseudoaleatoris, funcions hash o 'message authentication codes', entre d'altres elements criptogràfics.

A block cipher is an encryption algorithm that encrypts a fixed size of n-bits of data - known as a block - at one time. The usual sizes of each block are 64 bits, 128 bits, and 256 bits. So for example, a 64-bit block cipher will take in 64 bits of plaintext and encrypt it into 64 bits of ciphertext. In cases where bits of plaintext is shorter than the block size, padding schemes are called into play.

Exemple d'algoritme de flux:

XOR

Suposant que el missatge original és, en binari, aquest: $x_1x_2...x_n$, cada bit del missatge xifrat es calcularà així: $\mathbf{y_i = x_i \ XOR \ k_i}$, on k_i s'ha generat pel 'key stream generator' a partir d'un valor inicial aleatori (anomenat "seed") + una determinada funció matemàtica. Recordem la taula de veritat d'aquesta operació booleana (on es pot veure que, en definitiva, el resultat obtingut és 0 sempre que els operands valguin igual i 1 si valen diferent):

$\mathbf{y_i}$	$\mathbf{x_i}$	**XOR**	$\mathbf{k_i}$
0	0		0
1	0		1
1	1		0
0	1		1

La gràcia d'usar l'algoritme XOR és que no només la clau és la mateixa per xifrar i desxifrar (és un algoritme simètric) sino que el propi algoritme és el mateix en ambdues operacions: $\mathbf{x_i = y_i \ XOR \ k_i}$ A més, XOR no porta ròssec (i per tant, és computacionalment senzill i ràpid). D'altra banda, és difícil de trencar per força bruta si la clau és prou llarga però és fràgil davant l'anàlisi de freqüències .

A continuació es presenta un exemple de xifrat. Suposant que la seed sigui (k_1=1, k_2=0, k_3=0, k_4=0) -la qual no pot ser reusable!- i que la funció emprada és $k_{i+4}=k_i+k_{i+1}$ *(mod 2)* -la qual és molt insegura perquè és predecible!-, els següents k_i seran:

$k_5=k_1+k_2=1+0=1$
$k_6=k_2=k_3=0+0=0$
$k_7=k_3+k_4=0+0=0$
$k_8=k_4+k_5=0+1=1$
$k_9=k_5+k_6=1+0=1$

Per tant, el fluxe-clau és 100010011... Si el missatge original és, per exemple, 10101010, el missatge xifrat obtingut fent servir XOR serà, doncs:

x_i = 10101010
k_i = 100010011...
---
y_i = 00100011

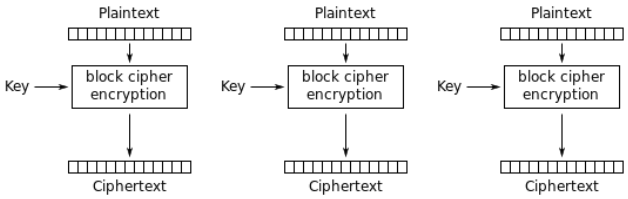
Modes d'operació en un algoritme de bloc genèric:

Suposant que el contingut d'un bloc del missatge original (l'anomenarem *m*) és, en binari, aquest: $x_1x_2...x_j$ (on *j* representa la longitud, en bits, del bloc), el contingut del bloc xifrat corresponent (l'anomenarem *c*) es calcula amb un determinat algoritme (com podria ser l'AES, però que aquí anomenarem genèricament "Encrypt"), així: **c=Encrypt(m,K)** on $c=y_1y_2...y_j$ i K representa la clau emprada, que serà d'una longitud en bits fixa però que no té perquè coincidir amb *j* . Per desxifrar es pot aplicar un altre algoritme, així: **m=Decrypt(c,K)**

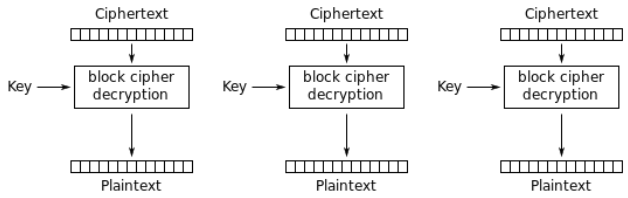
NOTA: Les longituds de les claus en AES poden ser 128 bits, 256 bits o més. No obstant, una longitud de 128 bits és a dia d'avui perfectament segura i longituds més llargues penalitzen en temps de computació

¿Què passa, però, si s'ha de xifrar més d'un bloc? Doncs que s'ha d'escollir un "mode d'operació", el qual determina com connectar els blocs. Depenent del mode d'operació escollit, la seguretat del missatge xifrat pot veure's compromesa encara que l'algoritme sigui òptim. Exemples de modes d'operació són:

ECB : Cada bloc es xifra independentment amb la mateixa K; molt insegur perquè es poden trobar patrons fàcilment



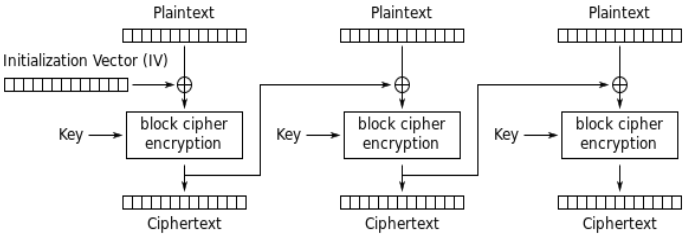
Electronic Codebook (ECB) mode encryption



Electronic Codebook (ECB) mode decryption

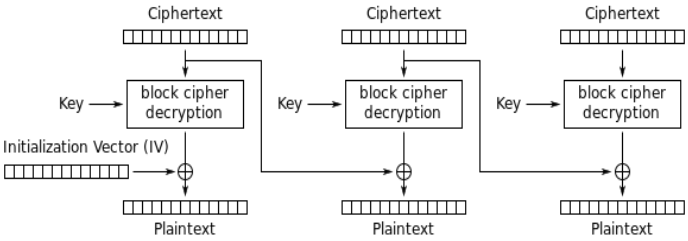
CBC : Es fa un XOR ($\oplus$) entre el bloc i un valor obtingut com a resultat extra de la funció Encrypt del bloc anterior. Si és el primer bloc, aquest valor llavors s'anomena "vector d'inicialització IV" i ha de ser generat aleatòriament per cada xifratge -amb el mateix tamany que un bloc- però no és pas secret -perquè se necessita pel desxifrat-. La idea és que el vector IV farà que cada missatge encriptat sigui únic, tot i que el missatge en clar sigui el mateix. El resultat d'aquest XOR és el que s'usa com a paràmetre d'Encrypt (a més de K) en comptes del bloc en sí. D'aquesta manera, el xifratge d'un bloc depén del del bloc anterior, fent que sigui un mode acceptablement segur. A continuació es mostra un diagrama explicatiu del procés de xifrat utilitzant aquest mode, el qual està obsolet degut a què recentment s'han trobat vulnerabilitats en el seu disseny

> NOTA: Respecte la generació aleatòria del vector IV (i, en general, de qualsevol tira de bits, com el "one-time pad" o la creació de qualsevol parell de claus) es podria parlar molta més estona perquè és un tema molt important: ¿com és possible generar un número aleatori a partir d'algoritmes concrets (i, per tant, reproduïbles)? La resposta és que no es pot i, per tant, entren en joc un conjunt d'elements com ara els **RNGs** ("Random Number Generators") hardware o software, molts dels quals solen recollir entropia ambiental (com ara els moviments del ratolí, la lectura analògica d'un sensor, etc) per combinar-la i obtenir un número el més aleatori possible. Per saber més es pot consultar https://www.random.org o https://en.wikipedia.org/wiki/Random_number_generation



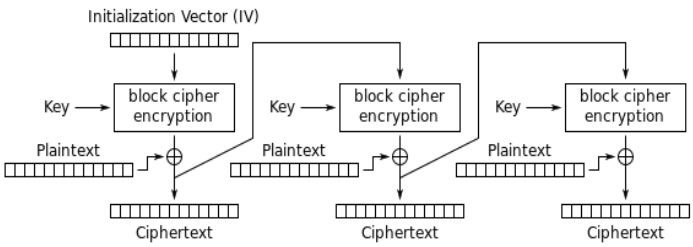
Cipher Block Chaining (CBC) mode encryption

El desxifratge es fa en diferent ordre. A continuació es mostra un diagrama explicatiu del procés de desxifrat utilitzant el mode CBC:

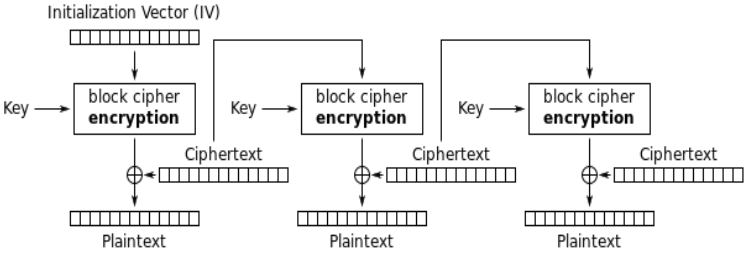


Cipher Block Chaining (CBC) mode decryption

CFB : Funciona com un algoritme de flux. El IV ha de ser no predecible. El desxifrat és la mateixa operació que xifratge

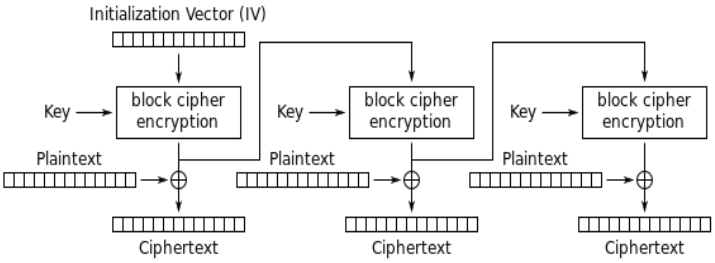


Cipher Feedback (CFB) mode encryption

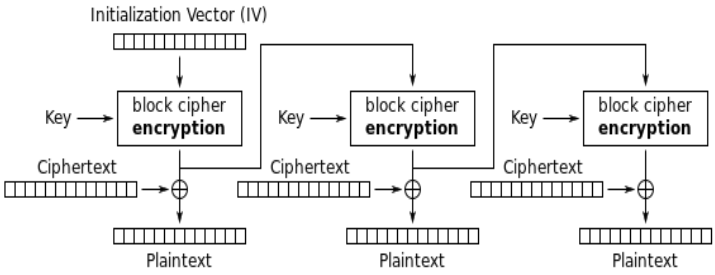


Cipher Feedback (CFB) mode decryption

OFB : Transforma algoritme de bloc en de flux. El IV ha de ser únic però no necessàriament aleatori

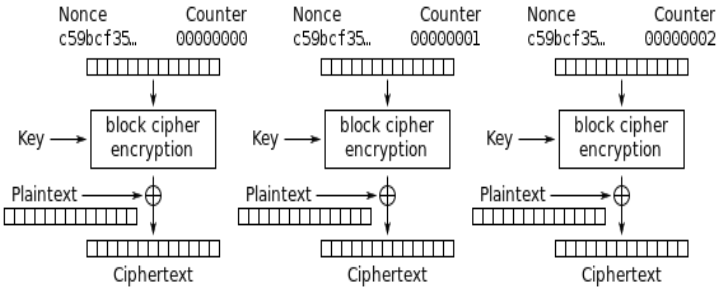


Output Feedback (OFB) mode encryption

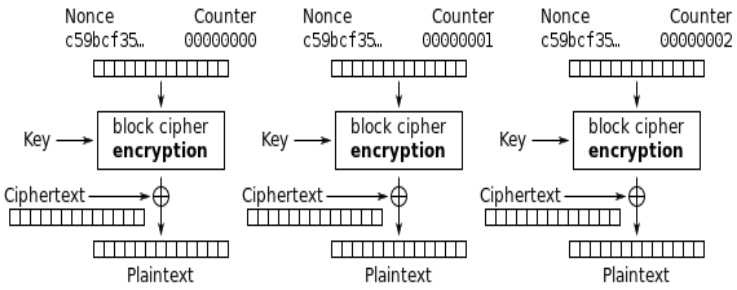


Output Feedback (OFB) mode decryption

CTR : Transforma algoritme de bloc en de flux. No s'usa IV sinò diferents "comptadors", els quals no han de ser secrets però sí únics



Counter (CTR) mode encryption



Counter (CTR) mode decryption

GCM : This block cipher mode has been widely adopted because of its great efficiency and performance. Moreover, GCM operation falls into the **AE** algorithm category; AE (from "Authenticated encryption") algorithms are designed to provide both data authenticity (integrity) and confidentiality In other words: an AE cipher provides message integrity in the symmetric algorithm itself, whereas non-authenticated ciphers need to rely on signed hashes for message integrity (as weÍl study in next pharagraph). Let's explain this: normalmente los cifrados simétricos solo proporcionan confidencialidad pero no proporcionan integridad -también llamada autenticidad de los datos- (es decir, no comprueban que los datos transferidos no hayan sido modificados). Por este motivo, es muy normal hacer uso de AES como cifrado para el canal de datos, y usar algún tipo de función hash (tal como veremos en los siguientes párrafos) para comprobar la integridad de los datos transferidos. No obstante, gracias a modos de operación de AES como el GCM, somos capaces de dotar a AES tanto de confidencialidad, como de integridad en la misma operación sin hacer uso de algoritmos adicionales y, además, con una única clave, sin necesidad de comprovar hashes externos.

> NOTA: **AEAD** (from "Authenticated Encryption with Associated Data") is a variant of AE that allows to check the integrity of both the encrypted and unencrypted information in a message. It is required, for example, by network packets: the header needs integrity but must be visible; payload, instead, needs integrity and also confidentiality. Both need authenticity.
>
> NOTA: TLS 1.3 only allows AEAD cipher suites, which means AES-GCM/AES-CCM and ChaCha20-Poly1305 are the only options available. Moreover, they are intended to improve performance and power consumption in devices with acceleration for AES

Una explicació molt més detallada sobre aquests modes d'operació (i molts d'altres com ara el OCB, el EAX o el CCM, etc) es troba als articles https://ctf101.org/cryptography/what-are-block-ciphers o també a https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Autenticació en criptografia simètrica:

Per pròpia definició de clau compartida, només els extrems que la coneixen poden establir una comunicació segura. Per tant, l'autenticació ja ve implícita (una altra cosa és que aquesta clau la robi algun "hacker"...però llavors estaríem en la mateixa situació que si es robés una clau privada a la criptografia asimètrica: tota l'estructura de seguretat se'n va en orris).

Integritat en criptografia simètrica (en algoritmes no AE/AEAD):

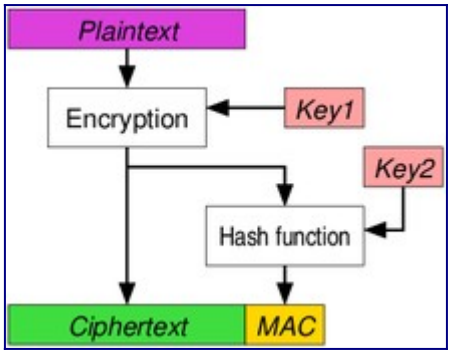
El mecanisme més habitual (si l'algoritme emprat no és de tipus AE/AEAD) per detectar que un missatge no hagi sigut eventualment compromès durant una transmissió (és a dir, no hagi sigut manipulat) és calcular, abans d'enviar res, el "hash" del contingut del missatge i, llavors, enviar aquest missatge amb el "hash" adjuntat inclòs en ell . D'aquesta manera, en rebre el conjunt missatge+hash, el receptor recalcularà el hash a partir del contingut del missatge rebut i el compararà amb el valor del hash rebut: si són iguals, voldrà dir que el missatge original no ha sigut modificat; si no ho són, el missatge es rebutjarà. Per tant, en resum: si a la confidencialitat i autenticació que atorga l'ús de claus simètriques a més hi afegim l'ús de "hashes" per tal de proporcionar integritat als missatges, ja tenim totes les característiques necessàries que ha de complir una comunicació per ser considerada segura (excepte el no repudi!!).

Aquest plantejament, no obstant, té un problema: ¿com podem garantir que no hagi sigut manipulat el propi hash (o substituït per un altre després d'haver-se modificat malintencionadament el missatge original)? Doncs la solució és senzilla: xifrant-lo amb una clau simètrica (la mateixa o una altra): això garantirà que aquest "hash" ha sigut realment el generat a l'emissor, ja que és l'únic que coneix la clau (és a dir, garantirà la seva pròpia confidencialitat i autenticació)

> NOTA: If the hash is transmitted over a different, protected channel, it can also protect the message against modifications. This is sometimes be used with hashes of very big files (like ISO-images), where the hash itself is delivered over HTTPS, while the big file can be transmitted over an insecure channel.

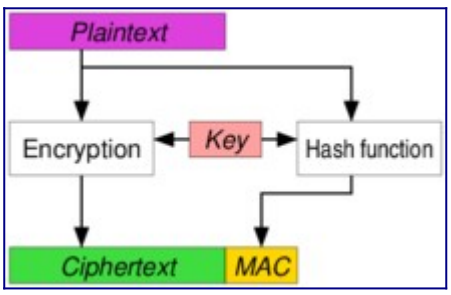
Així doncs, en el cas de fer servir criptografia simètrica per gaudir de confidencialitat, autenticació i integritat total, el procés és el següent (el diagrama de sota el mostra gràficament):

1.- L'emissor xifra el missatge que vol enviar amb la clau compartida.
2.- L'emissor calcula el "hash" del missatge xifrat però la funció hash escollida no fa el càlcul directament a partir només del missatge d'entrada (com faria una funció "hash" "clàssica") sinó que utilitza com a paràmetre addicional una clau (preferiblement diferent de la clau compartida emprada al punt 1, però que també ha de ser compartida amb el receptor...l'anomenarem "clau 2") que servirà per "autoxifrar" aquest "hash". A aquests tipus de "hashes autoxifrats" s'anomenen **HMAC**s (de "Hash Message Authentication Codes")
3.- L'emissor transmet el missatge xifrat + el HMAC de forma conjunta
4.- El receptor desxifrarà el HMAC amb la "clau 2" per obtenir el hash rebut en clar i recalcularà un nou hash per tal de comparar ambdós hashes.
5.- Si la comparació ha sigut exitosa, s'ha garantit la integritat del missatge i, per tant, el missatge és acceptat i el receptor procedeix a desxifrar el missatge fent servir la clau compartida

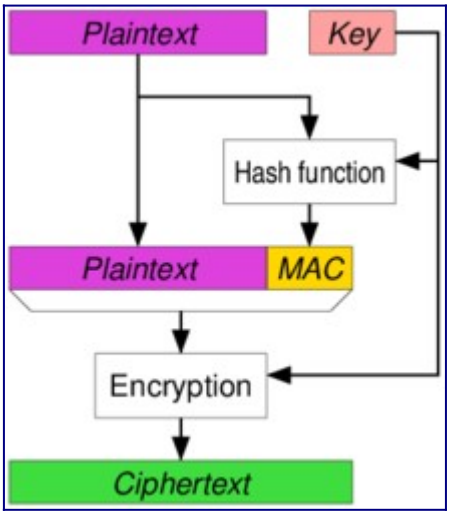


Tal com acabem de dir, les funcions que permeten generar HMACs, a diferència de les funcions "hash" "clàssiques" (emprades per emmagatzematr contrasenyes al fitxer "/etc/shadow" o per comprovar la integritat de fitxers amb les comandes *md5sum* o similars), no tenen com a únic paràmetre el missatge original a "hashear" sinó a més a més, tenen com a segon paràmetre la clau simètrica que empraran per xifrar el "hash" obtingut. D'algoritmes HMAC (també anomenats "keyed hash functions") n'hi ha uns quants i la majoria són adaptacions dels que ja coneixem quan vam veure els "hashes" "clàssics": SHA-1 (obsolet), SHA-2 (amb diferents tamanys de "hashes" resultants: **SHA256, SHA384...**), **SHA-3** (pertanyent a una altra "família" d'algoritmes anomenats genèricament Keccak), **Poly1305, BLAKE,** RIPEMD (obsolet),...

> NOTA: La seqüència de passes indicada és una anomenada "Encrypt-then-authenticate" (opció emprada, per exemple, per IPSec), que és la més segura (sense comptar els algoritmes AEAD) però no és l'única que existeix. Altres possibilitats serien "Authenticate-and-Encrypt" o "Authenticate-then-Encrypt". En la primera (opció emprada, per exemple, per SSHv1) es calcula el "hash" sobre el missatge_en_clar i, independentment, es xifra el missatge en clar amb la clau compartida, enviant-se llavors els dos elements ("hash" del missatge_en _clar i missatge xifrat) al receptor. En aquest cas només es necessitaria una sola clau (la compartida) per realitzar tot el procés. Al següent diagrama es pot veure gràficament:



En la segona (opció emprada, per exemple, a TLSv1.2) primer es calcula el "hash" sobre el missatge_ en_clar i després es xifra el conjunt "hash"+missatge_en_clar amb la clau compartida, enviant-se llavors aquest únic element xifrat en conjunt ("hash"+missatge_en_clar) al receptor. Igualment, aquí només es necessitaria una sola clau (la compartida) per realitzar tot el procés. Al següent diagrama es pot veure gràficament:



En qualsevol d'aquests casos, el receptor desxifrarà igualment el missatge rebut fent servir la mateixa clau compartida i recalcularà de nou el hash per tal de comparar-lo amb el valor rebut (l'ordre d'aquests dos pasos dependrà de si mètode utilitzat és "A&E" o "A-t-E"

Unfortunately, the 'generic composition' approach above is not the right answer for everyone: it can be a little bit complicated and, moreover, it requires you to implement two different primitives (say, a block cipher and a hash function for HMAC), which can be a hassle. Last, but *not* least, it isn't necessarily the fastest way to get your messages encrypted: the efficiency issue is particularly important if you're either working on a constrained device like an embedded system, *or* you're working on a fast device, but you just need to encrypt lots of data. (this is the case for network encryptors, which have to process data at line speeds, typically many gigabytes per second!). For all of these reasons, as we have already said, there are specialized block cipher modes of operation called AE/AEAD which handle both the encryption and the authentication in one go, usually with a single key.

Confidencialitat en criptografia asimètrica:

La idea bàsica de la criptografia asimètrica és, tal com ja s'ha comentat, que les dades que siguin xifrades amb una clau pública (la del receptor) només podran ser desxifrades amb la clau privada d'aquest mateix receptor. Els algoritmes de xifrat-desxifrat asimètric funcionen així, doncs:

1.- Receptor envia primer la seva clau pública (és a dir, coneguda per tothom) als eventuals emissors
2.- Emissor utilitza aquesta clau pública per xifrar el missatge
3.- Emissor envia el missatge a receptor
4.- Receptor utilitza la seva clau privada (és a dir, només coneguda per ell) per desxifrar el missatge

Tal com es pot veure, en aquest procediment mai es transmeten les claus privades i l'intercanvi de les claus públiques és senzill, ja que no cal que es transmetin utilitzant un canal segur. Per tant, el gran avantatge de la criptografia de clau pública respecte la criptografia simètrica és evident: la primera no necessita que es transmeti prèviament cap clau secreta per un canal insegur entre l'emissor i el receptor, resolent-se així el problema de l'ou i la gallina.

NOTA: Per resoldre el problema de que un extrem hagi d'enviar una clau a un destí per un mitjà insegur, a més de la criptografia asimètrica, han aparegut altres solucions, com ara el protocol Kerberos. En aquest protocol apareix el concepte de "servidor de distribució de claus" (KDC), que actua com a intermediari de confiança entre l'emissor i el receptor. Bàsicament el procés de comunicació és així: l'emisor i el KDC han de tenir compartida prèviament entre ells una clau secreta (KEK); igualment el receptor i el KDC també han de tenir compartida prèviament entre ells una altra clau secreta (la manera de com aconseguir això queda fora de l'àmbit d'aquesta nota). Cada cop que l'emissor vulgui enviar un missatge al receptor, demanarà una clau simètrica aleatòria "d'usar-i-llençar" al KDC, el qual la generarà i la distribuirà tant a l'emissor com al receptor, xifrada amb la seva respectiva KEK per garantir la seguretat d'aquesta clau. Lògicament, si una KEK és compromesa, el "hacker" es podrà fer passar per l'emisor (o el receptor, segons el cas). I si és el propi KDC el comprometés, tot el sistema de seguretat estarà comprometés.

Al seu torn, l'inconvenient principal de la criptografia de clau pública respecte la criptografia simètrica és el consum de recursos: per xifrar/desxifrar la primera és molt més costosa en temps de CPU (és a dir, la criptografia simètrica és més eficient computacionalment). Concretament, mentre els algoritmes simètrics es basen en confusió i difusió d'informació a partir de la repetició d'iteracions d'una funció simple, els algoritmes asimètrics es basen en problemes de teoria matemàtica de nombres i solen ser expressats en forma d'equacions.

D'altra banda, en el cas de la criptografia simètrica a mesura que augmenta el número de persones (l'anomenarem *n*) que es vulguin comunicar de forma segura entre sí,el número de claus augmenta seguint la fòrmula $n \cdot (n-1)/2$, cosa que no passa amb els algoritmes de clau pública, on només hi ha $2 \cdot n$ claus.

L'algoritme asimètric actualment més utilitzat per xifrar/desxifrar és el RSA.

Concepte de criptografia híbrida:

La criptografia híbrida combina el sistema de clau compartida i el de clau pública per obtenir tots els seus avantatges sense els seus inconvenients. En aquest sistema només s'utilitza la criptografia de clau pública (recordem, molt exigent computacionalment) per transmetre al receptor únicament una clau compartida (i no pas tot el missatge, que seria prohibitiu). Aquesta clau compartida normalment serà generada de manera dinàmica i efímera per aquesta comunicació (per tant, un "hacker" que esbrini la clau compartida emprada en una determinada comunicació no la podrà fer servir per cap més altra comunicació, ni passada ni futura). Una vegada s'hagi enviat aquesta clau compartida mitjançant el canal segur establert per la criptografia de clau pública, s'utilitzarà a partir de llavors la criptografia simètrica (molt més ràpida i molt menys costosa computacionalment) per xifrar el gros del missatge. Per tant, en resum, la criptografia híbrida combina la conveniència del sistema de clau pública i l'eficiència del sistema de clau compartida.

> NOTA: Amb la criptografia híbrida també es gaudeix de no-repudi (cosa que no passava amb la criptografia simètrica només) ja que s'evita que es pugui compartir la mateixa clau simètrica per més d'un remitent.

D'entre els protocols possibles per realitzar aquesta negociació prèvia per l'intercanvi segur de claus en un canal insegur i de manera no autenticada, destaca el protocol Diffie-Hellman (també anomenat simplement "DH" o "DHE" de "ephemeral") i les seves variants de corba elíptica ("ECDHE"), que és el protocol que utilitza, per exemple, el programa SSH o també la tecnologia TLS. Un vídeo il.lustratiu que explica molt bé com funciona el protocol DH és: https://www.khanacademy.org/computing/computer-science/cryptography/modern-crypt/v/diffie-hellman-key-exchange-part-2

> NOTA: Antigament es feien servir altres protocols de clau pública per l'intercanvi de claus en un sistema híbrid (com ara el propi RSA) però amb el temps s'han anat descartant perquè no proporcionen "**forward secrecy**". This concept basically means that an adversary can record the encrypted conversation between the client and the server and later (if it is able to obtain the RSA private key) all the recorded conversations can be decrypted. So if an algorithm offers "forward secrecy", it'll give assurances that session keys will not be compromised even if the private key of the server is compromised. In other words: "forward secrecy" protects past sessions against future compromises of secret keys. The way to achieve that is by generating a unique session key for every session a user initiates, so the compromise of a single session key will not affect any data other than that exchanged in the specific session protected by that particular key.

Concepte de certificat:

Ja sigui fent servir la criptografia de clau pública per xifrar tot el missatge o bé només per establir de forma segura la clau simètrica que es faria servir per aquest xifrat, tenim un problema: cal establir l'autenticitat de la clau pública del receptor per tal de què l'emissor envïi el missatge a qui creu que li envia i evitar, així, impostors i atacs "MitM".

Una solució a aquest problema són els certificats. Un certificat simplement és una clau pública (és a dir, a la pràctica, un fitxer) que ha estat signada internament per un tercer (l'anomenat "Certificate Authority" o CA) que garanteix ("certifica") que el receptor és qui diu que és. Tal com estudiarem aviat, "signar" un document (en aquest cas, una clau pública) implica fer servir una clau privada pròpia i, per tant, una signatura determinada només la pot realitzar un determinat agent (en aquest cas, una CA). Si l'emissor manté un magatzem de claus públiques de diverses CA podrà comprovar que la signatura associada a un determinat certificat es correspongui efectivament a alguna d'aquestes CA "confiables"; si és així, el certificat llavors es donarà com a vàlid i es procedirà al l'enviament segur del missatge.

En el caso particular, por ejemplo, de un servidor web seguro (HTTPS), el certificado que ofrece a los navegadores cuando un usuario visita su web garantiza que dicho servidor es realmente quien dice ser (siempre y cuando su certificado venga firmado por una autoridad de certificación confiable por el navegador). Tal como se ha dicho, los navegadores ya vienen de serie con una lista de claves públicas reconocidas de varias autoridades de certificación confiables (los llamados "certificados raíz"...son certificados porque están a su vez firmados por la propia

CA para evitar suplantaciones) que permiten validar cualquier certificado del mundo firmado por cualquiera de las CA presentes en esa lista. En el caso de que visitemos un servidor HTTPS que ofrezca al navegador un certificado firmado por una CA no reconocida es cuando se nos mostrará un mensaje de advertencia, ya que este hecho puede ser sospechoso. De hecho, el navegador comprueba varias cosas para dar por válida la conexión y comenzar la comunicación encriptada:

-El certificado obtenido no haya expirado ya
-El certificado obtenido ha sido creado para el mismo nombre DNS del servidor web al que se está accediendo
-El certificado está firmado por alguien en el que se confía (lo que hemos comentado)

El format intern dels certificats pot variar, encara que el més habitual és el format UIT-T X.509v3. Un certificat amb aquest format sol tenir l'extensió ".crt" i està estructurat internament en vàries seccions internes, com ara:

*Número de sèrie
*Data de creació i d'expiració
*Nom i dades burocràtiques identificatives de l'entitat certificada, incloent el domini DNS que tingui registrat
*Còpia de la clau pública de l'entitat certificada
*Signatura de l'autoritat certificadora

NOTA: A més del X.509, existeixen altres formats, com ara:

*.p12: Corresponde al estándar PKCS7#12 que define un formato de fichero habitual para almacenar claves privadas juntas con su correspondiente certificado, protegido por un PIN
*.pem: Formato que desarrollo específicamente en su momento para el uso de certificados con correo electrónico. Actualmente también se usa para distribución de claves privadas.
*.cer: Formato muy frecuente para la distribución de certificados X.509. Es típico que una autoridad de certificación distribuya su certificado raíz con este formato.
*.p7b: Formato de estructura de firma electrónica PKCS#7, pero que no embebe el documento electrónico firmado, solamente el certificado y/o lista de certificados revocados.
*.key: Formato para la distribución de claves privadas.

Cualquier individuo o institución puede generar certificados digitales y convertirse así en una autoridad de certificación, pero si no es reconocido por quienes interactúen con el certificado, el valor del mismo es prácticamente nulo (servirá para hacer pruebas o para un grupo de usuarios que confíen en nuestra máquina). Por ello las autoridades de certificación deben ser reconocidas oficialmente como institución certificadoras de forma que su firma pueda ser reconocida como fiable, transmitiendo esa fiabilidad a los certificados emitidos por la citada institución.

NOTA: A nivel legal, las administraciones públicas que otorgan validez y oficialidad a las autoridades de certificación son la Fàbrica Nacional de Moneda y Timbre, el Ministerio de Industria, Turismo y Comercio o la Agència Catalana de Certificació, entre otras.

Por eso lo más habitual como administradores de sistemas informáticos será simplemente enviar nuestra clave pública (o más técnicamente, un fichero específico llamaod "solicitud de certificación" que incluye dicha clave pública) a una autoridad de certificación oficial para que sea ésta la que firme nuestra clave y nos devuelva el certificado generado pertinente que nos permita implementar, por ejemplo, servidores seguros sin que los clientes tengan problemas de confianza. Antiguamente solo existían CAs comerciales, las cuales cobraban una cierta cantidad de dinero por realizar la firma de nuestra clave pública; ejemplos son las empresas Thawte, Comodo, Verisign, Ancert, GlobeSSL o RSA. Afortunadamente, desde hace un tiempo existen además CAs que realizan este proceso gratuitamente, tales como Cacert (http://www.cacert.org) y, sobre todo, Let's Encrypt (https://letsencrypt.org). El único requisito que necesitan estas CAs gratuitas (como cualquier otra CA, de hecho) es que dispongamos de un dominio DNS registrado en Internet, ya que el vínculo entre el certificado y el propietario del mismo lo realizan a través de este dato.

NOTA: Les CAs poden realitzar més tasques a més de generar certificats a partir de sol.licituds, com ara renovar-los quan s'arribi a la seva data d'expiració, desactivar-los o fins i tot revocar-los definitivament si hi hagués algun problema (com per exemple el robatori de la clau privada usada per signar) mitjançant o bé el mètode CRL o bé el OCSP.

Les sigles "PKI" (de "Public Key Infrastructure") sovint s'utilitzen per referir-se a tot el necessari (tant a nivell de hardware com de software com de gestió respecte les CAs) per mantenir les comunicacions segures mitjançant l´ús de certificats i signatures digitals. Cal tenir en compte que el sistema PKI no és perfecte; les vulnerabilitats més importants que pot patir són dues: que un malware alteri a la nostra màquina client el depòsit de certificats de forma que tinguem importats certificats signats per CAs fraudulentes (això es fa sovint manualment per realitzar atacs de tipus "MitM" amb eines com Bettercap o similars) i que un atac als servidors d'una CA robi la seva clau privada, podent així l'atacant firmar les claus públiques de servidors fraudulents que no ho semblarien.

Autenticació en criptografia asimètrica:

Gràcies a les signatures digitals tant l'emissor com el receptor d'un missatge poden comprovar l'autenticitat de l'altra entitat. Les signatures digitals són un concepte pertanyent a la criptografia asimètrica: l'emissor utilitza la seva clau privada per signar el missatge i el receptor comprovar aquesta signatura ("verifica") amb la clau pública de l'emissor, prèviament obtinguda. És a dir, el procés bàsicament seria el següent:

1.- L'emissor signa el missatge a enviar amb la seva clau privada (que no coneix ningú més)
2.- (Opcionalment, l'emissor també pot xifrar el missatge a enviar fent servir la clau pública del receptor)
3.- L'emissor envia el missatge a receptor
4.- El receptor utilitza la clau pública de l'emissor per verificar la signatura i, per tant, assegurar-se que és ell, efectivament, qui l'ha enviat.
5.-(Opcionalment, si el missatge a més de signat ve xifrat, el receptor faria servir la seva pròpia clau privada per desxifrar-lo)

Existeixen diversos algoritmes asimètrics especialitzats en la creació/verificació de signatures. Per exemple tenim de nou el RSA (sovint implementat amb el format PKCS#1, sobre tot molt utilitzat per xifrar/desxifrar, com ja s'ha dit), el DSA, el ElGamal, el DHE (sovint implementat amb el format PKCS#3, usat sobre tot per intercanviar claus simètriques), les variants de corba el.líptica dels anteriors algoritmes (com ara ECDSA, ECDHE...tots amb el format PKCS#13, menys exigents computacionalment i energèticament), etc.

Integritat en criptografia asimètrica:

A l'igual que passava amb la criptografia simètrica, a la criptografia asimètrica també s'usen "hashes" per detectar que un missatge no hagi sigut eventualment manipulat durant una transmissió. El procediment és molt semblant: també es calcula, abans d'enviar res, el "hash" del contingut del missatge i llavors s'envia aquest missatge amb el "hash" adjuntat inclòs en ell. D'aquesta manera, en rebre el conjunt missatge+hash, el receptor recalcularà el hash a partir del contingut del missatge rebut i el compararà amb el valor del hash rebut: si són iguals, voldrà dir que el missatge original no ha sigut modificat; si no ho són, el missatge es rebutjarà.

Aquest plantejament, no obstant, té d'entrada el mateix problema que vam veure a l'apartat sobre integritat a la criptografia simètrica: ¿com podem garantir que no hagi sigut manipulat/substituït el propi hash? Doncs la solució en aquest cas no és igual però sí molt similar: signant-lo amb la clau privada de l'emissor, ja que això garantirà que aquest "hash" ha sigut realment el generat a l'emissor (l'únic que poseeix la seva clau privada). D'aquesta manera, si el "hash" d'un missatge ve signat, el receptor haurà de verificar aquesta signatura (amb la clau pública de l'emissor, recordem) i això implica que si el "hash" signat hagués sigut manipulat, no en podria fer la verificació perquè la signatura no quadraria. En conclusió: les signatures digitals, a més de l'autenticitat, ens permeten garantir també la integritat del missatge aportada pels "hashes".

Cal dir que a la pràctica, i a diferència de l'explicat a l'apartat anterior, en realitat només se sol signar el "hash" (que és una tira de bits relativament curta, la qual també rep sovint el nom de "fingerprint") i no pas el missatge sencer. Això és perquè realitzar (i verificar) una signatura és un procés computacionalment molt costós, essent aquest fet més evident com més extens sigui el contingut a signar. I fent-ho així no es perd cap característica de seguretat (ni autenticació ni integritat).

NOTA: També existeixen els "fingerprints" de les claus públiques, que no són més que el resultat d'haver-les-hi aplicat un "hash". Aquests "fingerprints" serveixen per identificar una determinada clau pública de forma ràpida i còmoda per tal de facilitar la seva gestió. Per exemple, si una clau RSA típica pot tenir una llargària de 2048 bits, el seu "fingerprint" SHA-2 o MD5 serà només de 256 bits.

Resumint, el procés complet que permet una comunicació confidencial+autenticada+amb integritat seria:

1.- L'emissor calcula el hash del missatge que vol enviar
2.- L'emissor signa el hash amb la seva clau privada i adjunta aquest hash signat amb el missatge
3.- L'emissor fa servir la clau pública del receptor per xifrar el missatge (normalment incloent el hash signat)
4.- Es transmet el missatge xifrat i signat.
5.- El receptor desxifrarà el missatge fent servir la pròpia clau privada.
6.- El receptor comprovarà la signatura del hash fent servir la clau pública de l'emissor.
7.- El receptor recalcularà el hash a partir del missatge rebut i el compararà amb el hash rebut ja verificat
8.- Si la comparació és correcta, el missatge és acceptat