09/02/2022

Cifratura

Abbiamo due grandi categorie di cifratura:

- Cifrature basate su sistemi di **sostituzione** e **trasposizione**. Sono usate principalmente nelle cifrature di tipo **simmetrico**.
- Cifrature basate su sistemi **matematici**. Sono usate principalmente nelle cifrature **asimmetriche**.

Cifratura simmetrica

Viene usata la **stessa chiave** sia per cifrare, che per decifrare.

Cifratura asimmetrica

Vengono usate **due chiavi diverse**: se ne uso una per cifrare, uso l'altra per decifrare. Quindi entrambe possono essere usate per entrambe le cose, ma la stessa non può sia cifrare che decifrare.

Una chiave viene chiamata **pubblica**, e l'altra **privata**.

La chiave **pubblica** può essere resa nota al mondo (è pubblica, duh), e viene usata per **cifrare** il messaggio.
Quella **privata** si usa per **decifrarlo**, e va tenuta **segreta**.

Nei sistemi a cifratura asimmetrica, chiunque può cifrare un messaggio usando la chiave pubblica, ma solo il proprietario, che ha la chiave privata, può decifrarlo.

Data Encryption Standard

Storicamente, il primo algoritmo di cifrazione, **DES**, inventato nel 1977, ha le caratteristiche di avere:

- Block cypher dalla dimensione di 64 bit
- Usa una chiave di 56 bit

Nel 1977 andava bene, ad oggi no. Mediamente si rompe in 1 ora.

3DES

Il DES ma eseguito 3 volte. Lento.

Advanced Encryption Standard

L'**AES**, aggiudicato come standard da un bando della NIST, è buono.

17/02/2022

Se uso OFB e riutilizzo il vettore di inizializzazione, la cifratura diventa **vulnerabile** agli attacchi di tipo **known-plaintext**: l'attaccante conosce un plaintext P e il corrispondente ciphertext C.

Esempio

A partire da IV ⇒ OFB, calcola l'output stream O.
$C = O \oplus P$
$O \oplus P = O \oplus P \oplus P = O$
E quindi $C \oplus O = P$

Controesempio - 23/02/2022

Dato un noto IV, non deve essere prevedibile l'IV successivo.
Se IV è prevedibile, CBC è vulnerabile da attacchi di tipo **chosen-plaintext**.
L'attaccante ha la possibilità di far cifrare un plaintext P a sua scelta.

CBC: $C = E(k, P \oplus IV)$ supponiamo anche che P appartiene ad un insieme limitato di possibili valori (es. elezioni USA).

Opponente: $P' \oplus IV_{next} \Rightarrow C' = E(k\, P' \oplus IV \oplus IV_{next} \oplus IV_{next})$

Cifratura asimmetrica

Caratteristiche

- Esistono **2 chiavi differenti**, entrambe possono essere usate **sia per cifrare che per decifrare**, ma entrambe devono essere usate **in coppia**: se una la uso per cifrare, l'altra la uso per decifrare.
- Si basano su **algoritmi matematici**: computazionalmente è una crittografia più lenta.

Osservazioni

- **Non è** più sicura della cifratura simmetrica.
- L'unico parametro che definisce il grado di sicurezza è la **lunghezza della chiave**.
- Non rende la cifratura simmetrica obsoleta, ma spesso vengono usate insieme.
- C'è il problema della **distribuzione certa** delle chiavi pubbliche: bisogna essere certi che la chiave pubblica appartenga effettivamente a X.

In questo schema di cifratura ogni utente deve:
- Generare le sue due chiavi, una pubblica (e quindi distribuita) e l'altra privata.
- Possedere una "raccolta" delle chiavi pubbliche degli altri utenti con cui deve comunicare.

Nel 1976 Diffie ed Hellman hanno postulato i requisiti di un algoritmo di cifratura a chiave pubblica.
Hanno detto che un buon algoritmo deve funzionare così:
- Deve essere facile, per un utente A, generare le sue chiavi PUB_A e PK_A.

- Deve essere facile, per B, cifrare un messaggio M con PUB_A : C = E(PUB_A, M)
- Deve essere facile, per A, decifrare C : M' = D(PK_A, C) = M
- Deve essere impossibile (mooolto difficile), per un opponente, ricavare PK_A da PUB_A.
- Deve essere impossibile (“), per un opponente, noto C e PUB_A, ricavare M.

Nota: C = ciphertext, M = messaggio

Due soli anni dopo è stato pubblicato l'algoritmo di cifratura che è tuttora lo standard: l'**algoritmo RSA**, che soddisfa tutti i criteri.

Questo algoritmo si basa su una metodologia che implementa l'**aritmetica modulare**.

Aritmetica modulare

Definizione di **congruenza**: dati m, n ∈ $\mathbb{N}$, diremo che m è congruente ad n modulo p, p ∈ $\mathbb{N}$ se: $m \,\%\, p = n \,\%\, p$, e si scrive m ≡ n modulo p.

Quindi sono congruenti due numeri che, divisi per un terzo numero, danno lo stesso resto.

Un'altra definizione è la seguente: se n = m % p, allora io posso scrivere m = q * p + n.
(q = quoziente, che non mi interessa).
Ora considero l ≡ n modulo p ⇒ l = k * p + n
A questo punto faccio la differenza m - l.

24/02/2022

Piccolo teorema di Fermat

Consideriamo:
- **p** ∈ $\mathbb{N}$, primo
- **a** ∈ $\mathbb{N}$, non multiplo di p

Consideriamo i termini: a, 2a, 3a, …, (p-1)a

Questi p - 1 termini sono sicuramente tutti **non congruenti** tra di loro.
Calcolando mod p per ciascuno di loro si ottengono **tutti i possibili resti** 1, 2, … (p - 1).

Per le proprietà del prodotto:
a * 2a * 3a * … * (p - 1) = 1 * 2 * 3 * … * (p - 1) ⇒ a^{p-1}(p - 1)! ≡ (p - 1)! mod p

La loro differenza deve essere multiplo di p
⇒ a^{p-1}(p - 1)! - (p - 1)! = k * p
⇒ (p - 1)! * (a^{p-1} - 1) = k * p
(p - 1)! deve essere multiplo di p
(a^{p-1} - 1) deve essere multiplo di p
⇒ a^{p-1} ≡ 1 mod p

Il piccolo teorema di Fermat può essere generalizzato al caso in cui **non** si fa l'ipotesi iniziale che *p* sia primo, e in questo si chiama **teorema di Eulero**.

Teorema di Eulero

Consideriamo:
- **p** ∈ ℕ
- **a** ∈ ℕ, non multiplo di p

Considero i **k** termini $\mathbf{a_i}$ tali che $\mathbf{1 \leq a_i \leq p - 1}$.

Considero i k termini che si ottengono moltiplicando ciascuno degli $\mathbf{a_i} * \mathbf{a}$:
$a_1 * a, a_2 * a, \ldots, a_k * a$

Questi termini:
- Sono **non congruenti** tra loro.
- Sono congruenti ad uno dei possibili termini $a_1, a_{2,}, \ldots, a_k$

Quindi:

$a_1 * a, a_2 * a, \ldots, a_k * a = \prod_{i=1}^{k} a_i * a \equiv \prod_{i=1}^{k} a_i \bmod p$

$\Rightarrow a^k * \prod_{i=1}^{k} a_i \equiv \prod_{i=1}^{k} a_i \bmod p$

la loro differenza deve essere multiplo di p.

$\Rightarrow a^k * \prod_{i=1}^{k} a_i - \prod_{i=1}^{k} a_i = l * p$

$\Rightarrow \prod_{i=1}^{k} a_i * (a^k - 1) = l * p$

$\prod_{i=1}^{k} a_i$, $(a^k - 1)$ non possono essere multipli di p.

k è il numero di predecessori di p che sono primi rispetto a p.
k è una funzione di p, si chiama **funzione toziente** e si indica φ(p).

$\Rightarrow a^{\varphi(p)} \equiv 1 \bmod p$

02/03/2022

Il toziente ha alcune proprietà interessanti.

1. Se p è un numero primo $\Rightarrow \varphi(p) = p - 1$
2. Se p è un numero primo $\Rightarrow \varphi(p^k) = p^k - p^{k-1}$
3. Se p, q sono due numeri primi tra di loro $\Rightarrow \varphi(p * q) = \varphi(p) * \varphi(q)$

Usando le proprietà di $\varphi(p) \Rightarrow$

1. **Calcolare il toziente** di un numero qualunque, di cui si conosce la decomposizione in fattori primi.

Es. $\varphi(360) = 3^2 * 2^2 * 5 * 2 = 2^3 * 3^2 * 5$
$\Rightarrow \varphi(2^3 * 3^2 * 5) = \varphi(2^3) * \varphi(3^2) * \varphi(5)$

2. Calcolare il modulo di potenze

Es. $4^{100003} \, mod \, 33$

$\varphi(33) = \varphi(3 * 11) = \varphi(3) * \varphi(11) = 2 * 10 = 20$
$4^{100003} = 4^{100000} * 4^3 = (4^{20})^{5000} * 4^3$

$\Rightarrow (4^{20})^{5000} * 4^3 \, mod \, 33 =$
$\Rightarrow (4^{20})^{5000} \, mod \, 33 * 4^3 \, mod \, 33$

Algoritmo RSA

Prevede diverse fasi.

Generazione delle chiavi

- Scegliere p, q primi, → ∞
- Determinare n = p * q
- Calcolare φ(n) = φ(n)*φ(q) = (p - 1) * (q - 1)
- Scegliere un valore e t.c. 1 < e < n, primo rispetto a φ(n), non è richiesto che e → ∞, typ: e = 65537. (e, n) → **chiave pubblica**.
- Determinare un ulteriore valore *d* t.c. e * d ≡ 1 mod φ(n). (*d*, n) → **chiave privata**.

Calcolo di *d* - primo metodo

$$d = e^{\varphi[\varphi(n)]-1} \, mod \, \varphi(n)$$

Dimostrazione

$$e * d = e * e^{\varphi[\varphi(n)]-1} = e^{\varphi[\varphi(n)]} \, mod \, \varphi(n) = 1 \Rightarrow e * d = e \equiv 1 \, mod \, \varphi(n)$$

Calcolo di *d* - teorema di Euclide

Il teorema di Euclide, solitamente, serve per calcolare il **massimo comun divisore**.
g = gcd(a, b) con a, b ∈ Z

Il teorema esteso, invece, permette di calcolare, oltre a g, anche i valori x, y che soddisfano l'**identità di Bezout**: a * x + b * y = g.
gcdex(a, b) → g, x, y

I tre valori *x*, *y* e *g* sono calcolati dal teorema di Euclide esteso, dandogli in pasto i valori di input a, b.

Come valore di *a* scegliamo *e*, come valore di b scegliamo φ(n).
Sputerà fuori un valore *y* che a noi non serve, e un valore *g* che sarà **1**.

09/03/2022

Algoritmo RSA [continuazione]

Abbiamo visto che ci sono diverse fasi.

Generazione delle chiavi

- Si scelgono due numeri ***p*** e ***q*** grandi e primi
- Si calcola il loro prodotto $n = p * q$ e il toziente $\varphi(n) = (p - 1)(q - 1)$ del loro prodotto, facilmente calcolabile noti *p* e *q*.
- Si sceglie un numero ***e***, non necessariamente grande ma primo rispetto al toziente: questo è la **public key**.
- Si individua un ulteriore numero ***d*** tale che il prodotto $e * d$ è congruente a $1 \, mod \, \varphi(n)$, che costituisce la **private key**.

Calcolo di *d*

Si può usare un metodo adatto a farlo su carta e penna, o con uno computazionale.

Criptazione con public key

m = messaggio da cifrare (plain message).
m deve essere un numero, e deve essere **< *n***.
Se è una stringa, farò associare a ciascun carattere un valore numerico.

Il cypher message ***c*** si ottiene facendo $c = m^e \, mod \, n$, quindi viene usata la **public key**.

Decriptazione con private key

Se la criptazione l'ho fatta con la **public key**, la decriptazione la faccio con la **private key**.

Per ottenere di nuovo il messaggio originale (plain message) devo fare:
$m' = c^d \ mod \ n$
E questo ***m'*** deve essere uguale ad ***m***, cioè il messaggio iniziale.

Dimostrazione che m' = m

Se $m' = c^d \ mod \ n$, e $c = m^e$
Ottengo che $m' = m^{ed} \ mod \ n$

Ricordiamo che $e * d$, per definizione, è **congruente a** $1 \ mod \ \varphi(n)$.
$e * d \equiv 1 \ mod \ \varphi(n)$

Il che si può scrivere anche in quest'altra maniera:
$e * d = k * \varphi(n) + 1$

Riprendendo il calcolo originale, ottengo che
$m' = m^{k*\varphi(n)+1} \ mod \ n$

Che posso anche scrivere come
$m' = m^{k* \varphi(n)} * m \ mod \ n$

Che posso anche scrivere come
$m' = m^{k* \varphi(n)} \ mod \ n \ * \ m \ mod \ n$

Che posso anche scrivere come
$m' = (m^{\varphi(n)} \ mod \ n)^k \ * \ m \ mod \ n$

Notiamo che l'espressione $m^{\varphi(n)} \ mod \ n$ è il **teorema di Eulero**, quindi quello fa **1**.

1^k farà sempre 1, quindi diventerà
$m' = m \ mod \ n$

E siccome $m \ mod \ n$ è il resto della divisione tra ***m*** ed ***n***, ma abbiamo scelto un numero ***m*** **minore di *n***, allora $m \ mod \ n$ è uguale ad m.
$m' = m$

Spiegazione k

Se $10 \ mod \ 3 \ = \ 1$
Io posso dire che $10 \ = \ 3 * k + 1$
k è un valore fisso

Formule utili nei calcoli RSA

a = numero generico

k = numero generico intero

Nel caso di **potenze pari**,
$a^{2k} \bmod n = (a^2 \bmod n)^k \bmod n$

Nota: il modulo, essendo il resto, posso ripeterlo.

Nel caso di **potenze dispari**,
$a^{2k+1} \bmod n = (a^2 \bmod n)^k * a \bmod n$

Esempio di calcolo RSA

Generazione delle chiavi

Scelgo ***p*** e ***q***, primi e grandi (non troppo in questo caso d'esempio).
$p = 7, q = 13$
$n = 7 * 13 = 91$
$\varphi(n) = (p-1)(q-1) = 6 * 12 = 72$

$e = 5$, scelto da me (a caso) in quanto è primo rispetto a $\varphi(n) = 72$.

Con la formula dell'altro giorno, $d = e^{\varphi[(\varphi(n)]-1} \bmod \varphi(n)$

Devo fare il toziente di toziente.

Noi sappiamo che $\varphi[\varphi(n)]] = \varphi(72)$

Possiamo scomporre 72 in $2^3 * 3^2$, e ottenere
$\varphi(72) = \varphi(2^3 * 3^2) = (2^3 - 2^2)(3^2 - 3) = 4 * 6 = 24$

E quindi
$d = 5^{24-1} \bmod 72 = 5^{23} \bmod 72$

Con la formuletta di prima dell'esponente dispari
$d = (5^2 \bmod 72)^{11} * 5 \bmod 72$

Con la formuletta di prima dell'esponente dispari
$d = 25^{11} \bmod 72 * 5 \bmod 72$

Con la formuletta di prima dell'esponente dispari
$d = (25^2 \bmod 72)^5 * 25 * 5 \bmod 72$
Con la formuletta di prima dell'esponente dispari
$d = (49^5 \bmod 72)^5 * 25 * 5 \bmod 72$

Con la formuletta di prima dell'esponente dispari
$d = (49^2 \, mod \, 72)^2 * 49 * 25 * 5 \, mod \, 72$

Con la formuletta di prima dell'esponente dispari
$d = (25^2 \, mod \, 72) * 49 * 25 * 5 \, mod \, 72$

Calcoliamo che $(25^2 \, mod \, 72)$ fa 49, finendo il calcolo otteniamo che
$d = 29$

16/03/2022

Esempio

$n = 77 \, (p = 11, \, q = 7)$
$e = 43$
$c = 5$
$d = 7$
$m = ?$

$m = c^d \, mod \, n = 5^7 \, mod \, 77 = (5^2 \, mod \, 77)^3 =$
$= (25^2 \, mod \, 77) * 25 * 5 \, mod \, 77 = 9 * 25 * 5 \, mod \, 77 = 47$

Scambio di Diffie-Hellman

Algoritmo **asimmetrico** che permette di scambiarsi in modo sicuro **1 chiave simmetrica**.

Dobbiamo dare alcune definizioni preliminari.

Radice primitiva

Dato un numero ***p*** $\in \mathbb{N}$, si definisce **<u>radice primitiva</u>** di *p* un valore ***a*** che gode di questa proprietà: $a \, mod \, p, \; a^2 \, mod \, p, \; \ldots, \; a^{(p-1)} \, mod \, p \Rightarrow$ si ottengono tutti i possibili **resti** 1, 2, …, (p-1) in qualche ordine.

Logaritmo discreto

In particolare, dati ***p*** numero primo, e ***a*** radice primitiva di *p*, esiste uno ed un solo ***i*** tale che
$1 \leq b \leq p - 1 \Rightarrow \exists! \, i, \; 1 \leq i \leq p - 1 + i$ (*!* significa uno e uno solo)
$b = a^i \, mod \, p$, ***i*** viene detto **<u>logaritmo discreto</u>** base ***a***, mod ***p*** di ***b*** e si scrive
$i = d \, log_{a,p} b$

Funzionamento

Alice e Bob scelgono un numero primo ***p*** e una radice primitiva di *p* ***g***.
Questi numeri non devono essere segreti.

Una volta fatta questa scelta, Alice sceglie un valore ***a < p*** che tiene **segreto**, e con questo valore che ha scelto calcola $A = g^a \, mod \, p$.

Questo valore ***A*** che ha calcolato lo manda a Bob.

Bob fa la stessa cosa dall'altra parte: sceglie a sua volta un numero ***b < p*** che tiene segreto, calcola $B = g^b$ e manda questo numero ***B*** ad Alice.

Alice, ricevendo B, calcola $B^a \, mod \, p$.

Applicando la proprietà della potenza del modulo:
$B^a \, mod \, p = g^{ab} \, mod \, p = k$.

k sarà la **chiave simmetrica** che userà Alice per comunicare.

Bob farà la stessa cosa, riceve ***A*** e calcola $A^b \, mod \, p = g^{ab} \, mod \, p = k$

Esempio

Numero primo $p = 29$
Una radice primitiva di ***p*** $g = 8$

A. $a = 3 \Rightarrow A: 8^3 \, mod \, 29 = 19, B = 3 \Rightarrow 3^3 \, mod \, 29 = 27 = k$
B. $b = 11 \Rightarrow A = 19 \Rightarrow 19^{11} \, mod \, 29 = (19^2 \, mod \, 29)^5 * 19 \, mod \, 29$
$= (13^5 \, mod \, 29) * 19 \, mod \, 29 = 6 * 19 \, mod \, 29 = 27 = k$

Lo scambio di Diffie-Hellman non è sicuro davanti a un attacco **man in the middle**.

Mallory si mette tra Alice e Bob e intercetta Alice, fingendosi Bob, quindi riceve il *A* di Alice e manda la sua lettera grande *Z* a Bob come se fosse la *A* di Alice.
Bob manderà la sua *B* ad Alice, e Mallory manderà la sua *Z* ad Alice, la quale la crederà come la *B* di Bob.

24/03/2022

Hashing

Input **x** (blocco dati di qualsiasi dimensione) -> Hash **H** -> Output **H(x)=h**.

L'output ha **lunghezza fissa** ed è fortemente dipendente da x (se x cambia di un 1 bit, H(x) cambia moltissimo).

Proprietà dell'hashing

- Proprietà **one way**: se conosco h, deve essere impossibile determinare x tale che H(x) = h.
- Proprietà **collision resistant**: dato x tale che H(x) = h, deve essere impossibile trovare y!=x tale che H(y) = h.

Quindi deve essere *computazionalmente molto difficile* trovare l'input dall'hash in output, e deve essere *estremamente improbabile* avere lo stesso hash in output per input differenti.

Secure Hash Algorithm

Famiglia di algoritmi di hashing pubblicata dal NIST, tutti basati su una costruzione Merkle-Damgård.
Ne esistono varie famiglie: SHA-1, SHA-224, SHA-256, SHA-384- SHA-512.

SHA-512

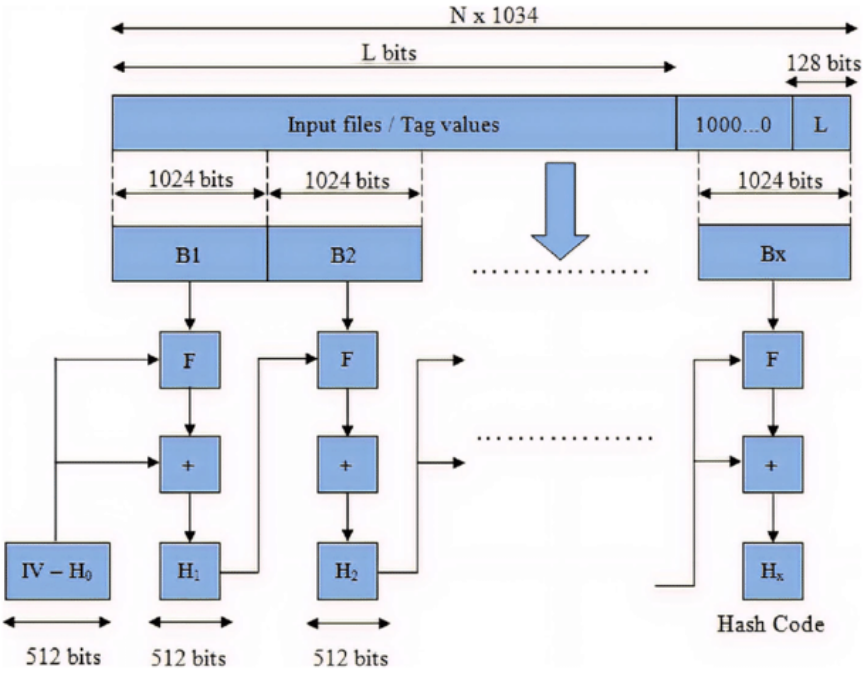
Si chiama così perché produce un hash di 512 bit.
Prevede in ingresso un messaggio di un massimo di 2^{128} bit, e lavora a blocchi di 1024 bit.

Per prima cosa aggiunge un **padding**, per una lunghezza tale che la lunghezza del messaggio originale **L sommato al padding modulo 1024** dia come resto **896**.

Poi ci aggiunge ancora **128 bit**, che descrivono la lunghezza del messaggio originale.

Inizializza un buffer da 512 bit formato da 8 registri a **64 bit** formati da **valori fissi**, che sono i primi 64 bit della parte decimale della radice quadrata dei primi otto numeri primi.

Il messaggio viene successivamente processato in blocchi da 1024 bit: assieme all'IV viene dato in pasto alla ***compression function***, il cui output viene XORrato con l'IV.

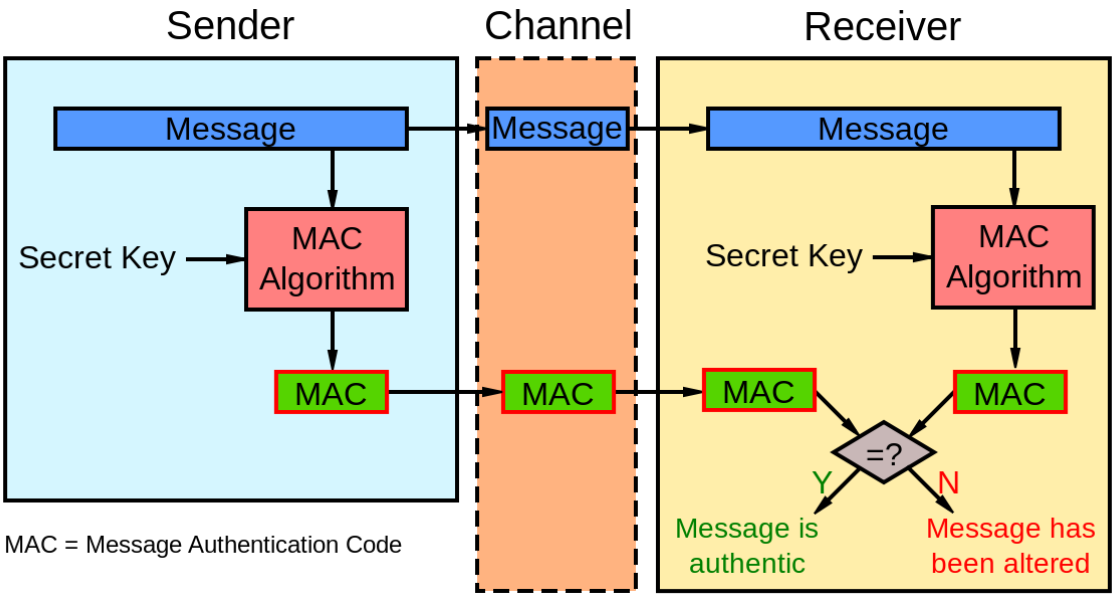


Alla password viene aggiunto un **salt**, una stringa casuale. Così se due utenti hanno la stessa password, l'hash sarà diverso.

L'altro grande campo dell'hash è l'**autenticazione dei messaggi**, una procedura che permette di verificare che un messaggio e la sua sorgente **non siano stati alterati**. Quindi verifica l'integrità e l'autenticità.

30/03/2022

L'autenticazione dei messaggi si può implementare con approcci diversi, ma tutti richiedono la generazione di un **tag** che si appende alla fine del messaggio.
Questo tag viene chiamato **Message Authentication Code**.



Implementazione MAC con cifratura simmetrica

Il sender, a partire dal messaggio originale, calcola l'**hash H**.
L'hash stesso viene **cifrato** con **chiave simmetrica e segreta** che naturalmente deve avere in possesso sia chi manda che chi riceve.
L'**hash cifrato** che esce è il MAC, che viene inviato assieme al messaggio.

Il receiver, per fare la verifica, per il messaggio ricevuto viene generato indipendentemente l'**hash H**.
Si prende il MAC che viene **decifrato**, e dalla decryption ottiene a sua volta il MAC.

Questo MAC calcolato viene confrontato con quello inviato: se uguale, il messaggio è autentico.

Implementazione MAC con cifratura asimmetrica

Il messaggio viene sottoposto al calcolo dell'hash, e viene cifrato con **chiave privata**. Si riceve il MAC, che viene inviato assieme al messaggio.

Il receiver, a sua volta, calcola l'hash del messaggio, decifra il MAC inviato usando la **chiave pubblica** e lo confronta con quello calcolato: se uguale, il messaggio è autentico.

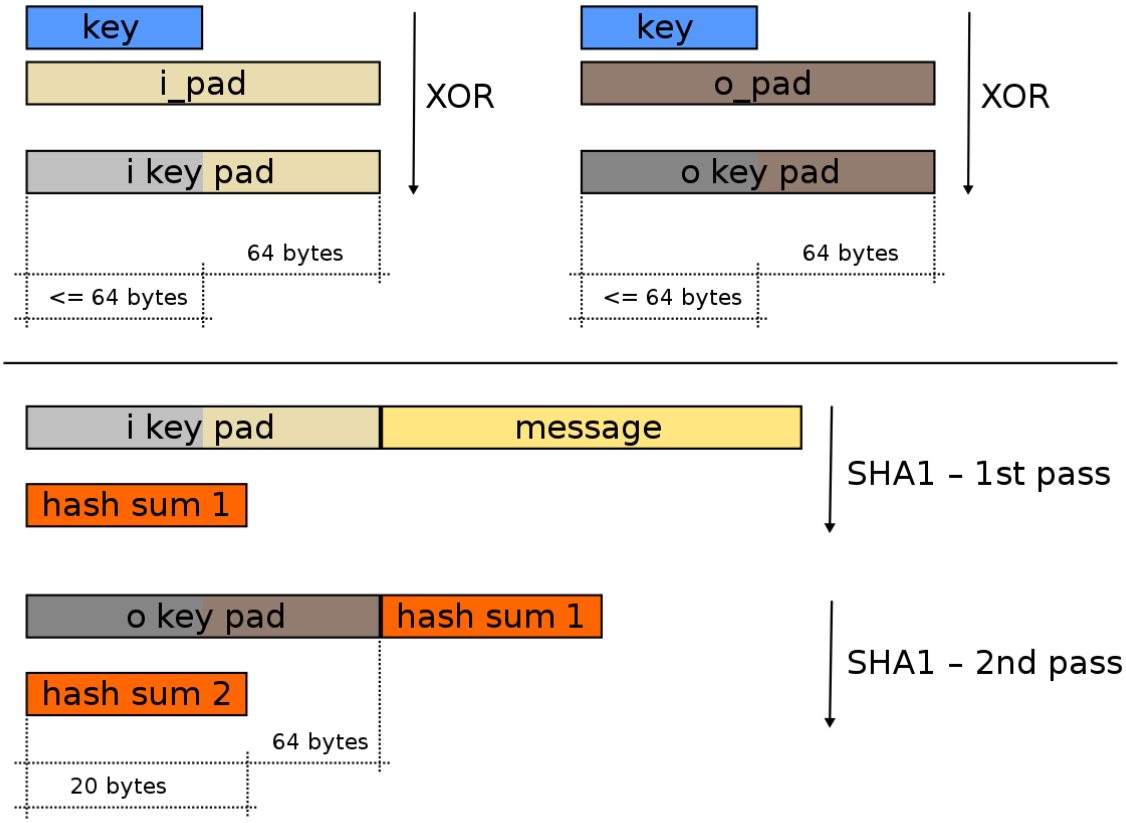
Implementazione MAC con valore segreto

Assieme al messaggio viene aggiunto un **valore segreto** che sia il sender che il receiver conoscono.
Il sender cifra il messaggio concatenato al valore segreto, e invia il messaggio con il valore ottenuto concatenato.

Il receiver calcola l'hash del messaggio assieme al valore segreto che già conosce, e confronta il valore calcolato con il valore ricevuto: se uguale, il messaggio è autentico.

HMAC

HMAC è una modalità per l'autenticazione di messaggi basata su una funzione di hash, utilizzata in diverse applicazioni legate alla sicurezza informatica. Tramite HMAC è infatti possibile garantire sia l'integrità, sia l'autenticità di un messaggio.



oiuhhuiooiuhhuuiooiuhuioiuhuiouhiouhioiohioooh

Firma digitale

Permette di:

- Verificare l'**origine** dei dati (autenticità)
- Che non siano stati **alterati** (integrità)
- Non repudiation - se firmo un documento non posso negare che non provenga da me

Per essere calcolata, viene calcolato l'**hash** del documento, che viene **cifrato con chiave privata**.
Il risultato viene agganciato al documento stesso.

Il messaggio firmato viene trasferito e per fare la verifica della firma bisogna fare le operazioni complementari: si prende il documento separatamente e si calcola l'hash.
Si prende la firma digitale e si decifra con la chiave pubblica, ottenendo l'hash inviato.
Se i due hash corrispondono, il messaggio è autentico.

Hashing su Python

Si può usare il modulo **hashlib**.

13/04/2022

La soluzione al problema del **controllo di appartenenza di una chiave pubblica** viene realizzata mediante un **documento** (certificato digitale) che attesta tale appartenenza **firmato da un ente ritenuto affidabile**.

Questo ente si chiama **certification authority**.

La CA distribuisce le chiavi pubbliche a tutti i soggetti che la utilizzano.

20/04/2022

Un certificato è composto da vari campi, tra cui la *firma digitale*.

11/05/2022 </covid>

Per richiedere un certificato il client deve:
- Generare una **coppia di chiavi** privata / pubblica.
- Inviare la chiave pubblica in un documento chiamato **Certificate Signing Request** (**CSR**).
- Esso lo invia alla **Certificate Authority** (**CA**).
- La CA **verifica i dati** del documento e del proprietario, per poi procede a rilasciare il certificato stesso al client.

Posso creare un certificato *self-signed* con:

```
openssl req -new -newkey rsa:4096 -x509 -sha256 -days 365 -nodes -out
nginx_cert.pem -keyout nginx_key.pem
```

Il browser controlla
- Che il certificato sia valido
- Che ti stai controllando veramente il server a cui ti stai collegando

Tipi di certificati

La stragrande maggioranza dei certificati sono i **DOmani Validation** (**DV**), che verificano l'appartenenza del dominio.
La validazione avviene in uno dei seguenti modi:
- Tramite **email**, che non è certamente la via maestra
- Via **HTTP**: l'hash del CSR è generato e dato all'*applicant*, che deve creare un file con il valore dell'hash e piazzarlo nel server web. Se il CA può riavere indietro questo file, il dominio è verificato.
- Via **DNS**: l'hash del CSR deve essere pubblicato in un record CNAME del dominio. Se il CA può riavere indietro questo record, il dominio è verificato.

Certificati revocati

Certificati che vengono **invalidati prima della loro naturale scadenza**.

12/05/2022

Protocollo TLS

In origine, quando è nato, si chiamava **SSL**.
Il protocollo TLS garantisce un canale sicuro tra due applicazioni comunicanti, con le ***AAA*** della sicurezza informatica:

- Confidenzialità: nessuno può vedere il contenuto del messaggio
- Integrità: i dati ritornano non alterati
- Autenticati: generalmente uno dei due (solitamente il server) deve essere autenticato

Il TSL si piazza tra l'application layer (HTTP) e il transport layer (TCP).
A sua volta, il TLS ha due stati:

- Sotto, il **TLS Record Protocol**: definisce il formato dei pacchetti TLS.
- Sopra, altri protocolli usati dal TLS.

Protocollo handshake

Client e server devono mettersi d'accordo su una serie di cose. Questo avviene nell'**handshake**.

Per prima cosa, tra client e server si deve conseguire una connessione TCP.

Una volta che la connessione TCP sia stabilita, il client deve mandare il primo messaggio chiamato ***ClientHello***, ancora in chiaro, mandando al server una serie di informazioni, come la versione del TLS, una lista di tutti gli algoritmi di cifratura che conosce, il metodo per lo scambio sicuro della chiave, algoritmo MAC (Message Authentication Code)… Il client provvederà anche una stringa chiamata ***client_random***.

Il server riceve questo messaggio *ClientHello* e decide quale algoritmo di cifratura utilizzare, mandandolo indietro al client assieme ad una stringa ***server_random*** in un pacchetto ***ServerHello***.

Immediatamente dopo aver inviato il *ServerHello*, il server manda anche il suo **certificato X.509**, e, per terminare questa sequenza di scambio, invia un ***ServerHelloDone***, con cui comunica di aver dato tutto al client e di star aspettando una risposta da parte sua.

Il client, ora, deve **verificare il certificato**, quindi controllare che sia valido, che il nome del proprietario corrisponda con il nome del server, ed eventualmente controllare che il certificato non sia stato revocato.
Se è tutto ok, il client invia un messaggio ***ClientKeyExchange***, con cui il client genera un'ulteriore sequenza segreta chiamata ***pre-master***, la cifra con la **chiave pubblica del server** (ottenuta dal certificato).

Il server invia un ***ChangeCipherSpec*** e un ***Finished***, che termina la fase di handshake e inizia la comunicazione cifrata tra client e server.

Calcolo chiave simmetrica

È un processo complesso e macchinoso, che da *client_random*, *pre_master* e *server_random* genera una chiave ***master_secret***.

Da *client_random*, *master_secret* e *server_random* viene generato un ***key block*** che conterrà 4 chiavi differenti e 2 vettori di inizializzazione.
Questo calcolo lo fanno indipendemtente client e server.

TLS Record Protocol

Una volta che l'handshake è finito, client e server possono iniziare a scambiare dati.
I dati sono inviati in **record**, il cui formato è definito dal **TLS Record Protocol**.

C'è un primo byte che definisce il *Content-Type*, che può essere *0x14*, *0x15*, *0x16* o *0x17*.
Nei due byte successivi è indicata la versione del protocollo TLS che stiamo utilizzando (tipicamente la *1.3*).
Due byte ulteriori che indicano la lunghezza.
Per ultimo, il payload vero e proprio.

HTTPS (HTTP over TLS)

L'**HTTPS** non è altro che l'HTTP con le risposte date in pasto al TLS.