

# Crittografia

Leonardo Brugnano

2022/23

# Chapter 1

## Lezione 1 (19/09/2022)

### 1.1 Introduzione

La **crittografia** costituisce l'insieme dei metodi di cifratura di messaggi, mentre la **crittoanalisi** costituisce l'insieme dei metodi di interpretazione di messaggi crittati.

Se consideriamo MSG l'insieme dei messaggi *in chiaro* e CRI l'insieme dei *crittogrammi*, allora si ha che la cifratura è una funzione

$$C : MSG \rightarrow CRI$$

che mappa il messaggio  $m$  nel crittogramma  $c$ , mentre la decifrazione è la funzione

$$D : CRI \rightarrow MSG$$

la quale è, prevedibilmente, la **funzione inversa** di  $C$ . Questo implica

$$D(C(m)) = m$$

**Osservazione:** La funzione di cifratura  $C$  deve essere ovviamente biunivoca affinché sia invertibile. Ciò significa che ad ogni crittogramma deve corrispondere uno ed un solo messaggio in chiaro, e viceversa.

### 1.2 Esempi storici

Nel V secolo a.C. si usava scrivere il messaggio sulla testa di un corriere pelato per poi aspettare che gli ricrescessero i capelli prima di mandarlo al destinatario, il quale, conoscendo lo metodo, rasava di nuovo il corriere per leggere il messaggio.

Sempre nel V secolo a.C. gli spartano usavano lo *scitale*. Per ulteriori approfondimenti, cercalo su internet.

Si arriva così al primo metodo di crittazione che ha gettato le basi per la crittografia moderna: il **cifrario di Cesare**. Esso è considerato il primo metodo di crittografia moderno in quanto consiste in una funzione matematica vera e propria. Esso consiste nello shiftare ogni lettera del messaggio di tre posizioni avanti. Il destinatario del messaggio, conoscendo il metodo, shifta ogni lettera del crittogramma di tre posizioni indietro per ricomporre il messaggio originale.

Il problema è però che se un intruso viene a scoprire il funzionamento del processo, è in grado di decrittare tutti i crittogrammi e quindi il cifrario diventa inutile.

### 1.3 Cifrari a chiave segreta (simmetrici)

Il cifrario di Cesare è il classico esempio di cifrario per uso ristretto, per il quale l'efficacia della cifratura si affida alla segretezza del metodo di cifratura. Dal momento che il metodo diventa pubblico, chiunque può decifrare i crittogrammi!

Si capisce quindi che questo tipo di cifrari va bene fintanto che poche persone se ne servono e non per cose importanti.

Si capisce quindi che per avere un cifrario che chiunque può usare, la sua efficacia non deve risiedere nella segretezza del funzionamento, in quanto anche se non è pubblico, prima o poi i crittoanalisti arrivano a capire come funziona.

Perciò, i cifrari moderni funzionano tramite il concetto di **chiave**: il messaggio viene crittato in funzione della chiave data in input

$$C(m, k) = c$$

ed il crittogramma può essere correttamente decrittato solo utilizzando la stessa chiave

$$D(c, k) = m$$

In questo modo solo la chiave deve essere segreta (ai due soggetti che comunicano), mentre le regole di cifratura sono pubbliche.

**Osservazione:** Siccome la segretezza del messaggio cifrato dipende solo dalla chiave, il numero delle chiavi possibili deve essere abbastanza grande da rendere inutile il provarle tutte (**attacco esaurente**), dato che ci vorrebbe troppo tempo. Inoltre, la chiave deve essere scelta in modo casuale, altrimenti si fornisce ai crittoanalisti indizi su quale potrebbe essere.

#### 1.3.1 Attacchi crittoanalitici

Un attacco ha l'obiettivo di forzare la decrittazione di un messaggio cifrato per ottenere l'originale. In generale, un crittoanalista può avere un comportamento **passivo**, limitandosi ad ascoltare la comunicazione, oppure uno **attivo**, disturbando la comunicazione per scopi personali.

Attacchi passivi possono essere:

- Cipher Text Attack: il crittoanalista rileva una serie di crittogrammi e li usa per capire quale sia la chiave.
- Known Plain-Text Attack: il cri. conosce coppie di messaggi ed i relativi crittogrammi  $(m_1, c_1) \dots (m_n, c_n)$ .
- Chosen Plain-Text Attack: come su, ma le coppie le ottiene su messaggi in chiaro da lui scelti.
- Chosen Cipher-Text Attack: analogo a sopra per crittogrammi da lui scelti.
- Bruteforce Attack: il cri. prova tutte le chiavi possibili.

### 1.3.2 Cifrari perfetti

Esistono cifrari inattaccabili, che però sono estremamente lenti da usare. In essi, nessuna informazione relativa al messaggio può essere ricollegata al crittogramma. Essi usano per ogni messaggio una chiave casuale diversa lunga quanto il messaggio. Il problema è che chi riceve il crittogramma deve anch'esso conoscere la chiave, e questa informazione deve poter essere passata tramite un canale sicuro.

### 1.3.3 Cifrari dichiarati sicuri

Quasi per ogni cosa, si utilizzano di fatto cifrari dichiarati sicuri, cioè cifrari che **finora** non sono stati violati dagli esperti e per i quali risalire al messaggio senza chiave richiede un tempo **non polinomiale**.

#### AES

L'Advanced Encryption Standard è il cifrario simmetrico più usato al giorno d'oggi. Esso usa chiavi corte, generalmente di 128 o 256 bit. Esso funziona a blocchi, cioè il messaggio viene diviso in e decrittato con la stessa chiave a blocchi di lunghezza uniforme. Le chiavi vengono generate in automatico dai dispositivi comunicanti nel modo visto più avanti.

## 1.4 Cifrari a chiave pubblica (asimmetrici)

In un sistema a chiave pubblica, ogni utente possiede due chiavi, una **pubblica** ed una **privata**. Se Alice vuole inviare un messaggio a Bob, usa la chiave pubblica di Bob per crittarlo, quindi Bob userà quella privata per decrittarlo.

$$\begin{aligned} c &= C(m, k_{pub}) \\ m &= D(c, k_{priv}) \end{aligned}$$

**Osservazione:** In questo sistema, chiunque conosce la chiave pubblica può crittare il messaggio, ma solo chi ha la corrispondente chiave privata personale può decrittarlo. La forza di questo metodo è che la chiave privata non deve essere condivisa con nessuno!

#### RSA

Il sistema RSA è l'effettivo metodo di crittazione che implementa il modello a chiave pubblica.

**Osservazione:** L'RSA è utilizzato nell'AES per lo scambio iniziale delle chiavi segrete.

**Osservazione:** Il sistema a chiave pubblica ha gli svantaggi di essere molto più lento di quello simmetrico, oltre al fatto sono esposti per loro natura ad attacchi chosen plain-text.

# Chapter 2

## Lezione 2 (20-27/09/2022)

### 2.1 Rappresentazione matematica di oggetti

Un alfabeto è un insieme **finito** di simboli. In tale contesto, un oggetto è una sequenza ordinata di simboli. Se consideriamo  $d(s, N)$  la lunghezza dell'oggetto più lungo all'interno di un insieme di  $N$  oggetti con un alfabeto di cardinalità  $s$ , ci interessa minimizzare tale valore per motivi di spazio, efficienza e semplicità, quindi cerchiamo il  $d_{min}(s, N)$ .

**Osservazione:** Un metodo di rappresentazione è tanto migliore tanto più  $d(s, N)$  si avvicina a  $d_{min}(s, N)$ .

#### Rappresentazione unaria

Con un alfabeto di cardinalità 1, si capisce che  $d_{min}(s, N) = N$ , in quanto un oggetto è costituito da una sequenza di più istanze dello stesso unico simbolo.

#### Rappresentazione binaria

Con un alfabeto di cardinalità 2, abbiamo  $2^k$  sequenze di lunghezza  $k$ , quindi  $d_{min}(2, N) \approx \log_2 N$ . Inutile dire che tale rappresentazione è estremamente più efficiente di quella unaria.

#### Rappresentazione s-aria

Portando la rapp. binaria al caso più generale, arriviamo ad avere  $d_{min}(2, N) \approx \log_s N$  in una rappresentazione s-aria.

**Osservazione:** Si capisce che una rappresentazione deve avere cardinalità  $\geq 2$  per essere considerata efficiente.

### 2.2 Teoria della Calcolabilità

Sappiamo che l'insieme dei problemi si divide in non decidibili (problem per i quali non esiste un algoritmo risolutivo) e decidibili.

**Osservazione:** È certo che esistono problemi indecidibili in quanto l'insieme dei problemi non è numerabile, mentre quello degli algoritmi sì. Quindi esistono problemi per i quali sicuramente non esiste il corrispondente algoritmo risolutivo.

## 2.3 Teoria della Complessità

I problemi decidibili si dividono a loro volta in problemi trattabili (risolvibili in tempo polinomiale) ed intrattabili (costo esponenziale).

### Problemi trattabili

Nel caso di un problema risolvibile in tempo polinomiale, supponiamo che sia risolto in  $cn^s$  secondi dal computer A. In tal caso un computer B potente  $k$  volte più di A ottiene un miglioramento di un fattore **moltiplicativo**  $k^{1/s}$  nel tempo di calcolo. Questo significa che se il problema ha costo cubico e B è  $10^9$  volte più potente di un computer A, allora esso sarà  $10^3$  volte più veloce di A. Se il costo è lineare, B sarebbe  $10^9$  volte più veloce.

### Problemi intrattabili

Nel caso il problema abbia costo esponenziale, supponiamo che A risolva il problema in  $c2^n$  secondi. In tal caso un computer B potente  $k$  volte più di A ottiene un miglioramento di un fattore **additivo**  $\log_2 k$  nel tempo di calcolo. Ciò implica che anche se B è  $10^9$  volte più potente di A, i tempi di calcolo rimangono praticamente invariati!

**Osservazione:** Riassumendo, i problemi a costo esponenziale sono intrattabili perché anche un enorme miglioramento al sistema di calcolo porta un miglioramento quasi nullo agli effettivi tempi di calcolo.

### 2.3.1 Classi P, NP, PSPACE, EXP

Da adesso considereremo solamente i problemi **decisionali**, cioè il cui risultato è booleano.

La classe di problemi P racchiude l'insieme dei problemi risolvibili in tempo polinomiale.

La classe NP racchiude l'insieme dei problemi la cui verifica di un certificato richiede tempo polinomiale anche se la risoluzione del problema in sé non è polinomiale.

La classe PSPACE racchiude l'insieme dei problemi che hanno costo in spazio polinomiale.

La classe EXP racchiude i problemi risolvibili in tempo esponenziale.

**Osservazione:** Valgono le seguenti proprietà insiemistiche:

$$P \subset NP \subset PSPACE \subset EXP$$

Anche se non si sa se le inclusioni siano proprie o no.

### 2.3.2 Problemi NP-completi

I problemi NP-completi possono essere logicamente definiti come i problemi più difficili all'interno della classe NP, anche se ovviamente questa non è una definizione rigorosa.

**Osservazione:** Se un problema NP-completo si scoprissesse essere risolvibile in tempo polinomiale, allora tutti i problemi della classe NP lo sarebbero e quindi avremmo  $P = NP$ .

### Riduzione polinomiale

Un problema  $P_1$  si riduce in tempo polinomiale a un problema  $P_2$  ( $P_1 \leq_p P_2$ ) se esiste una funzione  $f$  calcolabile in tempo polinomiale tale che  $X$  è una soluzione di  $P_1$  se e solo se  $f(X)$  è una soluzione di  $P_2$ .

Conoscendo un algoritmo  $A$  di soluzione per  $P_2$  e la funzione  $f$  relativa alla coppia  $P_1, P_2$ , si può risolvere  $P_1$  trasformando ogni dato  $X$  di  $P_1$  in un dato  $f(X)$  di  $P_2$  e applicando l'algoritmo  $A$  a  $f(X)$ .

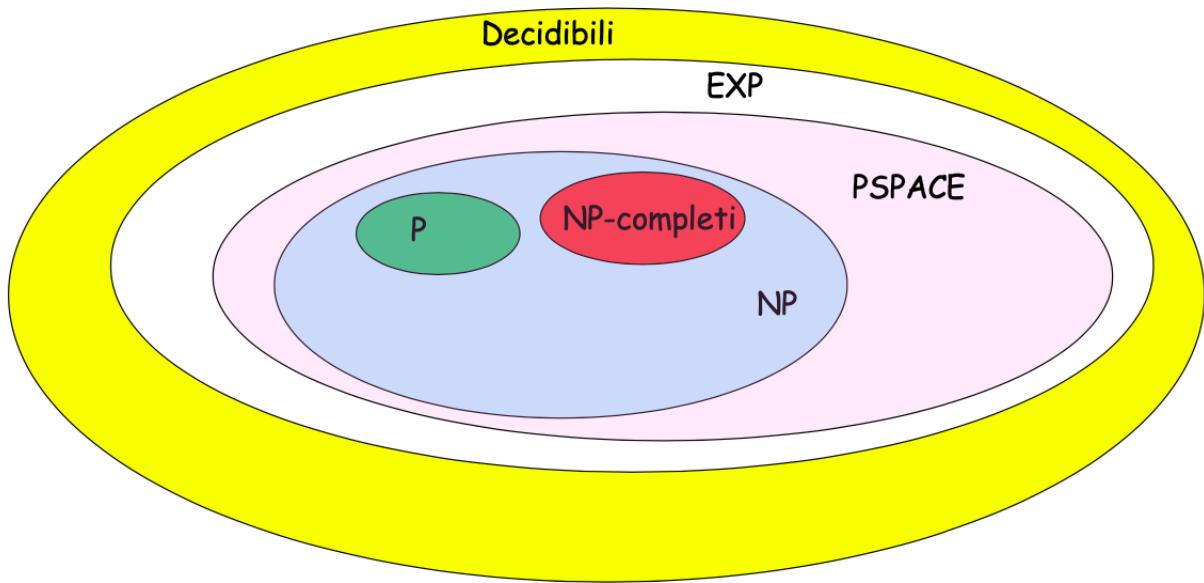
### Problema NP-arduo e NP-completo

Un problema  $P$  è **NP-arduo** se

$$\forall P' \in NP . P' \leq_p P$$

In tale contesto, si dice che un problema è invece **NP-completo** se è NP-arduo ed appartiene esso stesso ad NP.

### Schema riassuntivo



### 2.3.3 Classi co-P e co-NP

I problemi appartenenti alla classe co-NP (quindi anche co-P) certificano la **non esistenza** di una soluzione, a differenza degli altri che ne certificano l'esistenza.

Si ha ovviamente che  $\text{co-P} = P$ . Questo perché basta risolvere un problema in una delle due classi e poi complementare il risultato per trovare la soluzione al problema complementare nell'altra classe.

Per quanto riguarda co-NP e NP, si congettura che queste classi di problemi siano diverse.

# Chapter 3

## Lezione 3 (27/09/2022)

### 3.1 Teoria algoritmica della casualità

Partiamo facendo un esempio pratico e poi passiamo alla teoria.

#### 3.1.1 Esempio

Supponiamo di avere due sequenze di venti bit l'una:

- $h = 111\dots11$
- $h' = 1010011\dots01$

La probabilità di avere  $h$  lanciando una moneta 0/1 venti volte è  $1/2^{20}$ . Stessa per la seconda sequenza. Insomma, queste due sequenze hanno la stessa probabilità di uscire. Tuttavia, la prima non ci appare casuale e la seconda sì, dato che la prima possiamo descriverla velocemente individuando una struttura precisa. Lo stesso non vale per la seconda.

Definiamo a questo punto due algoritmi di generazione,  $A_h$  e  $A_{h'}$ , rispettivamente per le sequenze  $h$  e  $h'$ . Inoltre generalizziamo le due sequenze per portarle a lunghezza  $n$  generica. Quindi avremo  $h$  sequenza di  $n$  volte 1, e  $h'$  sequenza di  $n$  cifre, ognuna 1 o 0. Definiamo inoltre la **cardinalità** di un algoritmo la lunghezza della sua rappresentazione binaria.

Detto ciò, abbiamo che  $|A_h| = \theta(\log_2 n) + O(1)$ , con il primo fattore la quantità di bit che servono per rappresentare  $n$  in binario ed il secondo fattore la lunghezza del resto dell'algoritmo.

Per quanto riguarda  $A_{h'}$ , la sua cardinalità è  $|A_{h'}| = \theta(n)$ , dato che siccome la sequenza non ha una struttura precisa, bisogna descriverla bit per bit e quindi deve essere interamente memorizzata nell'algoritmo, il quale non fa altro che restituirla. Questo per  $h$  non succede, in quanto  $A_h$  può semplicemente stampare  $n$  volte 1 e tanti saluti, non dovendo perciò memorizzare l'intera stringa.

Concludendo, consideriamo casuali le sequenze binarie che NON ammettono un algoritmo di generazione la cui rappresentazione binaria sia più corta della sequenza stessa, anche se diamo una definizione formale di casualità più tardi.

#### 3.1.2 Sistemi di calcolo

I sistemi di calcolo sono numerabili, quindi possiamo riferirci ad essi con  $S_1, S_2$ , etc.

Consideriamo il sistema di calcolo  $S_i$  e  $p$  un programma per  $S_i$  che genera la sequenza  $h$ ; quindi abbiamo  $S_i(p) = h$ .

### Complessità di Kolmogorov

Definiamo complessità di Kolmogorov del sistema  $S_i$  la quantità

$$K_{S_i}(h) = \min\{|p| : S_i(p) = h\}$$

quindi essa è la lunghezza del programma più corto che genera  $h$  nel sistema  $S_i$ .

### Sistema universale

Il sistema di calcolo universale  $S_u$  è una particolare sistema capace di simulare tutti gli altri, quindi abbiamo

$$S_u(i, p) = S_i(p) = h$$

quindi la coppia  $q = \langle i, p \rangle$  è un programma che genera  $h$  in  $S_u$ , per cui vale

$$|q| = |i| + |p| = |p| + \theta(\log_2 i) = |p| + C_i$$

perciò la lunghezza di  $q$  dipende da  $i$  ma NON dalla sequenza  $h$ .

In generale abbiamo quindi

$$K_{S_u}(h) \leq K_{S_i}(h) + C_i$$

in cui l'ugualanza vale per le sequenze generate simulando  $S_i$ , non avendo programmi più brevi per  $S_u$ . Invece, la minoranza stretta vale per sequenze che si possono generare in  $S_u$  con programmi più brevi, simulando sistemi di calcolo diversi da  $S_i$ .

**Osservazione:** Tralasciando la costante  $C_i$  (trascurabile) la complessità di Kolmogorov del sistema universale costituisce un limite inferiore per la complessità di Kolmogorov di ogni altro sistema di calcolo.

**Definizione:** La complessità di Kolmogorov della sequenza  $h$  è  $K(h) = K_{S_u}(h)$ .

### Sequenze casuali e loro esistenza

Una sequenza è casuale secondo Kolmogorov se  $K(h) \geq |h| - \lceil \log_2 |h| \rceil$ . Questa è una definizione, non c'è nulla da ricavare o capire. Il fattore logaritmico sta solo perché altrimenti la definizione sarebbe stata troppo severa.

Inoltre, secondo Kolmogorov vale che  $\forall n . \text{ esistono sequenze casuali di tale lunghezza.}$

**Dimostrazione:** Il numero  $S$  di sequenze lunghe  $n$  è  $2^n$ , mentre il numero  $T$  di quelle non casuali tra queste. Inoltre, il numero  $M$  di sequenze lunghe al massimo  $n - \log_2 n$  è  $\sum_{i=0}^{n-\log_2 n-1} (2^i) = 2^{n-\log_2 n} - 1$ .

Queste  $M$  sequenze contengono anche tutti i programmi che generano le  $T$  sequenze non casuali, quindi  $T \leq M < S !!!$

Si conclude quindi che  $T < S$  e perciò esistono sequenze casuali di lunghezza  $n$ .

**Osservazione:** Nell'insieme di sequenze lunghe  $n$ , quelle casuali sono l'estrema maggioranza.

## Chapter 4

# Lezione 4 (03/10/2022)

Abbiamo visto che secondo Kolmogorov tra le sequenze lunghe  $n$ , quelle casuali sono la maggioranza. Questa è una buona notizia per il mondo della crittografia, tuttavia la cattiva notizia è che, **data una sequenza  $h$ , stabilire se essa è casuale secondo Kolmogorov è un problema indecidibile.**

**Dimostrazione:** Prendiamo un ipotetico algoritmo  $\text{Random}(h)$  che determina in tempo polinomiale se la sequenza  $h$  è casuale ( $\text{Random}(h) = 1$ ) o no ( $\text{Random}(h) = 0$ ).

A partire da esso, possiamo allora costruire un altro algoritmo  $\text{Paradosso}()$  che genera tutte le sequenze binarie in ordine di lunghezza crescente e, controllandone di ognuna la casualità con  $\text{Random}$ , termina quando trova la prima sequenza casuale  $h$  tale che  $|h| - \lceil \log_2 |h| \rceil > |P|$ , con  $P$  la sequenza binaria che rappresenta il programma complessivo, dato da  $\text{Paradosso} + \text{Random}$ .

Quindi quello che abbiamo è una cosa del genere:

Function  $\text{Paradosso}:$

```
for binary  $h \leftarrow 1$  to  $\infty$  do
    if ( $|h| - \lceil \log_2 |h| \rceil > |P|$ ) and ( $\text{Random}(h)=1$ )
        then return  $h$ .
```

Poiché esistono sequenze casuali di ogni lunghezza, si incontrerà certamente una sequenza  $h$  per cui sono soddisfatte entrambe le condizioni dell'if, quindi  $\text{Paradosso}$  terminerà per forza. Tuttavia le due clausole sono in contraddizione tra loro. Infatti la prima indica che il programma rappresentato da  $P$  è “breve”, e poichè questo programma genera  $h$  la sequenza non è casuale; la seconda indica invece che  $h$  è casuale poiché  $\text{Random}(h) = 1$ .

Si conclude che l'algoritmo  $\text{Random}$  non può esistere, e quindi determinare in tempo finito se una sequenza è casuale è un problema indecidibile.

## 4.1 Generatori di numeri pseudo-casuali

Una sequenza binaria si considera casuale se:

- $P(0) = P(1) = \frac{1}{2}$
- la generazione di un bit è **indipendente** dai bit precedentemente generati.

Nella pratica è impossibile ottenere generatori di sequenze casuali, perciò ci si deve accontentare di generatori di **sequenze pseudocasuali**, cioè sequenze che seguono una certa logica tuttavia praticamente inaccessibile ad un occhio esterno.

**Osservazione:** I generatori che vedremo restituiscono **numeri**, non sequenze. Quando parliamo di **sequenze** ci riferiamo alla sequenza di numeri restituiti in esecuzioni consecutive dei generatori.

### 4.1.1 Generatore lineare

Il generatore lineare prende in input un **seme**, cioè un numero  $x_0$  scelto casualmente.

**Osservazione:** A seconda delle applicazioni, la scelta del seme può ricadere sull'utente oppure sul sistema.

Inoltre, il generatore possiede dei parametri interni  $a, b, m$  per cui l'algoritmo di generazione dei numeri è il seguente:

$$x_i = (ax_{i-1} + b) \bmod m$$

Il problema di questo algoritmo è che il valore del prossimo numero generato dipende chiaramente dal precedente, e quindi il generatore potrà ben che vada generare una permutazione dei numeri da 0 a  $m - 1$ , prima di entrare in un ciclo.

È quindi importante scegliere  $a, b, m$  in modo tale da avere una permutazione pseudocasuale lunga esattamente  $m$ . Otteniamo così le seguenti condizioni sui parametri:

- $MCD(b, m) = 1$
- $a - 1$  deve essere divisibile per ogni fattore primo di  $m$
- $a - 1$  deve essere multiplo di 4 se anche  $m$  lo è

Ebbene, se queste condizioni sono rispettate, allora il generatore produce una permutazione esattamente lunga  $m$ .

### Conversione a generatore binario

Si capisce per avere sequenze binarie pseudo-casuali non si può semplicemente porre  $m = 2$ , per ovvi motivi. Quello che quindi si fa è mantenere  $m$  comunque molto grande, e per ogni  $x_i$ , numero  $i$ -esimo della sequenza generata, lo divido per  $m$  e guardo la parità della prima cifra decimale: se è pari allora ho 1, altrimenti 0. In questo modo possiamo associare ad ogni numero generato una cifra che sia 0 o 1 mantendendo la pseudo-casualità della sequenza.

### 4.1.2 Generatore polinomiale

Il generatore polinomiale opera con il seguente algoritmo:

$$x_i = (a_1x_{i-1}^t + a_2x_{i-1}^{t-1} + \dots + a_tx_{i-1} + a_{t+1}) \bmod m$$

Ovviamente, per sequenza lunghe più di  $m$  ha lo stesso problema di ciclicità del generatore lineare. Tuttavia, l'algoritmo è più complesso, garantendo in teoria una pseudo-casualità più forte.

Nella pratica, entrambi questi generatori (lineare e polinomiale) superano i seguenti 4 test statistici:

- test di frequenza
- poker test
- test di autocorrelazione
- run test

Il che significa che le sequenze da esse generati sono effettivamente pseudo-casuali e possono benissimo essere usate in ambito di testing. Tuttavia, questi generatori non vanno assolutamente bene in ambito crittografico, dato che per entrambi possono essere costruiti algoritmi di costo polinomiale che, analizzando sequenze prodotte da tali generatori, sono in grado di capire seme e parametri utilizzati!

## 4.2 Generatori crittograficamente sicuri

Si capisce che in ambito crittografico è importante usare generatori per i quali capire i parametri da essi utilizzati soltanto analizzando le sequenze prodotte **deve essere un problema intrattabile**.

A tal proposito, definiamo crittograficamente sicuri i generatori che superano il *test di prossimo bit*.

### Test di prossimo bit

Un generatore supera il test di prossimo bit se non esiste un algoritmo polinomiale in grado di prevedere con probabilità  $> \frac{1}{2}$  l' $(i+1)$ -esimo bit della sequenza a partire dalla conoscenza degli  $i$  bit già generati.

### Funzioni one-way

La costruzione di generatori crittograficamente sicuri si affida all'utilizzo di funzioni one-way.

Una funzione  $f$  si considera one-way se

- dato  $x$ , calcolare  $y = f(x)$  richiede tempo polinomiale
- dato  $y = f(x)$ , calcolare  $x$  richiede tempo esponenziale

Si capisce quindi che si può costruire generatori crittograficamente sicuro proprio a partire da funzioni one-way, in particolare impiegando specifici predicati **hard-core** di tali funzioni.

### Predicati hard-core

Un predicato  $b(x)$  è chiamato *hard-core* per una funzione one-way  $f(x)$  se

- dato  $x$ , calcolare  $b(x)$  è facile
- dato  $f(x)$ , calcolare od anche solo prevedere  $b(x)$  è difficile

In sostanza,  $b$  concentra in un solo bit la difficoltà computazionale di  $f$ .

### 4.2.1 Generatore BBS

Siano  $n = p * q$  il prodotto di due numeri primi grandi tali che

- $p \bmod 4 = 3$
- $q \bmod 4 = 3$
- $2\lfloor p/4 \rfloor + 1$  è primo
- $2\lfloor q/4 \rfloor + 1$  è primo

e  $x_0 = y^2 \bmod n$ , per qualche  $y$ .

Il generatore BBS impiega  $x_0$  come seme e calcola una successione  $\{x_i\}$  di interi lunga  $m \leq n$  secondo la legge

$$x_i = (x_{i-1})^2 \bmod n$$

Genera quindi una corrispondente sequenza binaria  $\{b_i\}$  secondo la legge

$$b_i = 1 \Leftrightarrow x_{m-i} \text{ pari}$$

Quindi il generatore BBS capovolge la sequenza binaria rispetto a quella di interi da cui è tratta. Questo è ciò che rende il generatore crittograficamente sicuro, facendogli passare il test di prossimo bit, proprio per le proprietà delle funzioni one-way.

# Chapter 5

## Lezione 5 (04/10/2022)

**Osservazione:** Il problema del generatore BBS è la sua lentezza, pur se opera in tempo polinomiale.

### 5.1 Generatori basati su cifrari simmetrici

I generatori basati su cifrari simmetrici lavorano a blocchi di bit e da essi producono parole di lunghezza costante. Essi sono crittograficamente sicuri e mediamente più veloci del BBS.

Qui vediamo il funzionamento di uno in particolare approvato come *Federal Information Processing Standard* negli USA, anche se i principi che segue sono comuni anche a tutti gli altri.

#### 5.1.1 FIPS

Siano

- $r$  la lunghezza in bit delle parole prodotte dal cifrario
- $m$  il numero di parole da produrre
- $s$  un seme casuale lungo  $r$  bit
- $k$  la chiave segreta del cifrario

**Osservazione:** Si nota subito che il generatore deve in totale produrre  $r * m$  bit.

#### Pseudocodice

```
Function Generatore( $s, m$ ):  
     $d \leftarrow$  rappresentazione su  $r$  bit di data e ora attuale;  
     $y \leftarrow \mathcal{C}(d, k);$   
     $z \leftarrow s;$   
    for  $i \leftarrow 1$  to  $m$  do  
         $x_i \leftarrow \mathcal{C}(y \text{ xor } z, k);$   
         $z \leftarrow \mathcal{C}(y \text{ xor } x_i, k);$   
        comunica all'esterno  $x_i.$ 
```

## 5.2 Test di primalità

### 5.2.1 Algoritmi randomizzati

Gli algoritmi randomizzati sfruttano una componente di casualità per aumentare le prestazioni al caso medio. Esistono due tipi di algoritmi randomizzati, quelli Las Vegas e quelli Monte Carlo.

#### Las Vegas

Gli algoritmi randomizzati Las Vegas sfruttano la casualità per aggirare condizioni sfavorevoli dei dati di input e generano **risultati sicuramente corretti in tempo probabilmente breve**. Un esempio classico è il *Quicksort*.

#### Monte Carlo

Gli algoritmi randomizzati Monte Carlo, a differenza dei Las Vegas, generano **risultati probabilmente corretti in tempo sicuramente breve**.

**Osservazione:** Si capisce quindi che questa seconda famiglia di algoritmi randomizzati viene usata quando è necessario porre un vincolo superiore ai tempi di calcolo, a costo che l'algoritmo talvolta si *sbagli*.

### 5.2.2 Test di Miller e Rabin

Il test di primalità di Miller e Rabin viene effettuato sfruttando un algoritmo randomizzato di tipo Monte Carlo. Infatti, il test può talvolta restituire risultati errati. Tuttavia, la probabilità di errore dell'algoritmo è direttamente influenzata dai parametri in input, di modo tale che essa possa essere arbitrariamente abbassata sino a farla diventare insignificante.

Il test di Miller e Rabin si basa su due proprietà dei numeri interi:

- $\forall N \text{ dispari} . N - 1 = 2^w * z$ , di cui la dimostrazione è immediata.

Per trovare  $w, z$  ci vuole tempo logaritmico sul valore di  $N - 1$ , cioè polinomiale sul numero di cifre di  $N - 1$ .

- $\forall N \text{ primo ed } y \in \{2, \dots, N - 1\}$ , chiamato **testimone**, valgono i seguenti predicati:

1.  $MCD(N, y) = 1$ , quindi  $y$  è coprimo di  $N$ .
2.  $y^z \bmod N = 1$  **OPPURE**  $\exists i \in \{0, \dots, w - 1\} : y^{2^i z} \bmod N = -1 (= N - 1 \bmod N)$

Il test consiste nel vedere se il numero da testare soddisfa entrambi i predicati. In tal caso viene considerato primo, anche se non è certo, dato che per i predicati non vale il *se e solo se*. Insomma, un numero può rispettare i predicati pur non essendo primo; tuttavia, l'insieme di questo tipo di numeri è relativamente piccolo, come afferma il lemma seguente.

#### Lemma di Miller e Rabin

Se  $N$  è composto, il numero di interi compresi tra 1 e  $N - 1$  che soddisfano i predicati P1 e P2 è strettamente minore di  $\frac{N}{4}$ .

Quindi, se  $N$  è composto, la probabilità di scegliere un testimone che soddisfa entrambi i predicati è strettamente minore di  $\frac{1}{4}$ . Si capisce che scegliendo a caso  $k$  testimoni, la probabilità che tutti soddisfino i predicati, restituendo perciò una risposta errata, è inferiore a  $\frac{1}{4^k}$ . Infatti, basta che uno solo non soddisfi i predicati per concludere che  $N$  non è primo.

**Esempio:** Già testando 10 testimoni, la probabilità di restituire un *falso primo* è minore di un milionesimo.

**Osservazione:** Ovviamente, non si può restituire un *falso composto*, dato che l'algoritmo può solo sbagliare dicendo che un numero è primo quando invece è composto, non il contrario.

### Pseudocodice

```
Verifica(N, y) // verifica se N è composto
    if (P1 == false OR P2 == false)
        return 1    // composto
    else
        return 0    // probabilmente primo (P = 1/4)

TestMR(N, k)
    for (i = 1; i <= k; i++)
        y = rand([2, N-1])
        if (Verifica(N, y) == 1)
            return 0    // N composto
    return 1    // N probabilmente primo (P = 1/(4^k))
```

# Chapter 6

## Lezione 6-7 (10-11/10/2022)

### 6.1 Analisi del Test di Miller e Rabin

Dato  $N$  dispari, i  $w$  e  $z$  tali che  $N - 1 = 2^w * z$ , con  $z$  dispari, si calcolano dimezzando  $N - 1$  per  $\theta(\log_2 N)$  volte. Questi due valori servono per calcolare il predicato P2.

Nella funzione  $Verifica(N, y)$ , il calcolo di P1 si esegue in tempo polinomiale con l'**algoritmo di Euclide**. Infatti, al caso pessimo, l'algoritmo opera in tempo lineare nel numero di cifre dell'input.

P2 si calcola invece con l'**algoritmo delle quadrature successive** in tempo polinomiale nel numero di cifre di  $N$ .

Facendo calcoli più approfonditi, si conclude che  $Verifica(N, y)$ , utilizzando l'algoritmo delle quadrature successive, ha complessità  $\theta(\log^3 N)$ , che è polinomiale nel numero di cifre di  $N$ .

**Nota:** L'algoritmo delle quadrature successive, nonché l'analisi del calcolo di P2, viene esplicato nel libro di testo. La spiegazione non è qui riportata.

Dato che  $TestMR(N, k)$  itera al più  $k$  volte, la complessità totale dell'algoritmo è  $O(k * \log^3 N)$ , che è quindi polinomiale!

### 6.2 Generazione di numeri primi

Ci poniamo ora il problema di generare numeri primi.

**Osservazione:** Questo interesse così grande per tali numeri, nasce dal fatto che sono alla base del funzionamento di molti cifrari.

Anche in questo caso, per la generazione vengono usati algoritmi randomizzati. In particolare, si può definire l'algoritmo *Primo* nel seguente modo:

**Function Primo( $n$ ):**

```

 $N \leftarrow 1S1$ , ove  $S$  è una sequenza di  $n - 2$  bit prodotti da un generatore
binario pseudo-casuale;
while  $\text{Test\_MR}(N) = 0$  do  $N \leftarrow N + 2$ ;
return  $N$ .

```

Come si può osservare, questo algoritmo genera un numero primo (binario) di almeno  $n$  bit. Il ragionamento è qui molto semplice: partendo da un numero dispari casuale di  $n$  bit, lo incremento finché non diventa primo.

Si potrebbe pensare che questo algoritmo abbia complessità elevata, ma la sua performance è garantita da una particolare proprietà dei numeri primi.

### Proprietà dei numeri primi

I numeri primi godono di una proprietà forte circa la loro densità sull'asse degli interi: il numero di interi primi e minori di  $N$  tende a  $N/\ln N$  per  $N \rightarrow \infty$ . Ciò significa che, se  $N$  è grande, in un suo intorno di lunghezza  $\ln N$  cade mediamente un numero primo.

Quindi, il numero di prove attese è polinomiale in  $n$  ( $= \dim N$ )

**Osservazione:** Si conclude che l'algoritmo  $\text{Primo}(n)$  ha complessità polinomiale in  $n$ .

## 6.3 Classe RP

Abbiamo già visto che l'esistenza di un certificato polinomiale *determina* l'appartenenza di un problema alla classe **NP**, ma che solo per la sottoclasse **P** sono noti algoritmi polinomiali per *determinare* un certificato, ovvero per risolvere il problema. L'introduzione dei **certificati probabilistici** (o **testimoni**) apre un modo nuovo di considerare i problemi.

Dato un problema decisionale  $\mathbf{P}$  ed  $x$  una sua istanza di input di lunghezza  $n$ , si dice che  $y$  è un certificato probabilistico di  $x$  se:

- $y$  ha lunghezza polinomiale in  $n$
- $y$  è estratto casualmente da un insieme associato ad  $x$

### Esempio (test di primalità)

Nel test di primalità,  $x = N$  e  $y \in \{2 \dots N - 1\}$  è scelto casualmente.

#### 6.3.1 Utilità del certificato probabilistico

Il certificato probabilistico è utile se esiste un algoritmo di verifica  $\mathbf{A}$  polinomiale in  $n$  ed applicabile ad ogni coppia  $(x, y)$ , che

- o attesti che  $x$  possiede la proprietà richiesta dal problema con probabilità  $> 1/2$
- oppure attesti con certezza che  $x$  non la possiede

**Osservazione:** Questo è proprio ciò che fa la funzione  $Verifica(N, y)$  nel test di Miller e Rabin.

Diciamo quindi che  $\mathbf{A}$  verifica  $\mathbf{P}$  in tempo polinomiale randomizzato.

### 6.3.2 Classe RP

La classe **RP** costituisce l'insieme dei problemi decisionali verificabili in tempo polinomiale randomizzato.

La relazione con le altre classi è la seguente:

$$\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$$

anche se l'eguaglianza è solo congetturata.

## 6.4 Cifrari storici

Cambiando completamente discorso, andiamo adesso vedere quali sono stati i più famosi cifrari nella storia.

### Principi di Bacone

Bacone stabilì dei principi essenziali per il buon funzionamento di qualunque cifrario. Essi sono:

- Le funzioni  $C$  (di cifratura) e  $D$  (di decifrazione) devono essere facili da calcolare.
- Deve essere impossibile ricavare la  $D$  se la  $C$  non è nota.
- Il crittogramma  $c = C(m)$  deve apparire innocente.

Dando uno sguardo approfondito ai vari cifrari storici, se ne distinguono due categorie:

- **Cifrari a sostituzione:** sostituiscono ogni lettera del testo in chiaro con una o più lettere dell'alfabeto secondo una regola prefissata.
- **Cifrari a trasposizione:** permutano le lettere del testo in chiaro secondo una regola prefissa.

La sostituzione può essere di due tipi:

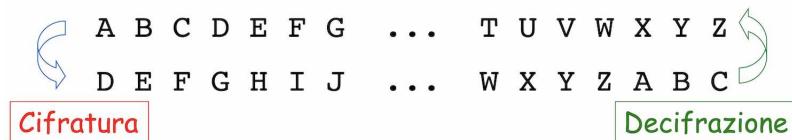
- **Sostituzione monoalfabetica:** alla stessa lettera del messaggio corrisponde sempre una stessa lettera nel crittogramma (e.g. cifrario di Cesare). Pur potendo impiegare funzioni di cifratura e decifrazione più complesse dell'addizione e della sottrazione in modulo, aumentando l'insieme delle chiavi, la sicurezza è comunque molto modesta.
- **Sostituzione polialfabetica:** alla stessa lettera del messaggio corrisponde una lettera scelta in un insieme di lettere possibili, secondo con una regola opportuna che tiene conto della posizione o del contesto in cui appare la lettera nel messaggio.

### 6.4.1 Cifrari a sostituzione monoalfabetica

#### Cifrario di Cesare

Il cifrario di cesare è il più antico cifrario di concezione moderna. Esso sfrutta una cifrazione monoalfabetica.

L'idea alla sua base è: **il crittogramma  $c$  è ottenuto dal messaggio in chiaro  $m$  sostituendo ogni lettera di  $m$  con quella tre posizioni più avanti nell'alfabeto.**



**ATTENTO A EVE**

**DWWHQWR D HYH**

I problema di questo cifrario è che **non ha una chiave segreta**, implicando che la segretezza del crittogramma dipende solo dalla conoscenza del metodo. Ciò rende questo cifrario utile solo per un uso ristretto, dato che se tutti ne conoscessero il funzionamento diventerebbe inutile.

#### Cifrario di Cesare generalizzato

Si può facilmente generalizzare il cifrario di Cesare introducendo una chiave segreta, che rappresenta il numero di posizioni da shiftare nell'alfabeto (originariamente sempre 3).

Tuttavia, anche questo cifrario è molto debole, dato che le chiavi possibili sono solo  $26!$ . Infatti, se una persona conosce il metodo, non ci mette tanto tempo a provarle tutte.

Insomma, questo cifrario è inutilizzabile ai fini crittografici. Inoltre, una sequenza di operazioni di cifratura e decifrazione può essere ridotta ad una sola operazione di cifratura o decifrazione. Quindi, comporre più cifrari non aumenta la sicurezza del sistema!

#### Cifrario affine

Un altro esempio di cifrario a sostituzione monoalfabetica è il cifrario affine, in cui una lettera in chiaro  $x$  viene sostituita con la lettera cifrata  $y$  che occupa nell'alfabeto la posizione

$$pos(y) = (a * pos(x) + b) \bmod 26$$

dove  $k = (a, b)$  è la chiave del cifrario. La relativa funzione di decifrazione è definita come:

$$pos(x) = a^{-1} * (pos(y) - b) \bmod 26$$

**Osservazione:** L'inverso di un intero  $a$  mod  $m$  esiste ed è unico se e solo se  $MCD(a, m) = 1$ . Di conseguenza, se  $MCD(a, 26) \neq 1$ , la funzione di cifratura non è iniettiva, e la decifrazione diventa impossibile. Di conseguenza, le chiavi possibili sono  $12 * 26 = 312$ , che sono troppo poche.

**Osservazione:** Se la segretezza dipende unicamente dalla chiave, allora:

- il numero delle chiavi deve essere così grande da essere praticamente immune da ogni brute-force attack.
- la chiave segreta deve essere scelta in modo causale.

### Cifrario completo

Ancora altro esempio di cifrario a sostituzione monoalfabetica, nel cifrario completo si prende una permutazione arbitraria dell'alfabeto come chiave. In tal modo, lettera in chiaro di posizione  $i$  viene sostituita con la lettera di posizione  $i$  nella permutazione.

ABCDEF <span style="font-size: 2em;">GHIJKL<span style="font-size: 1em;">MNOPQRSTUVWXYZ</span></span>
S <span style="font-size: 2em;">DTKB<span style="font-size: 1em;">JOHRZCUNYE<span style="font-size: 1.5em;">PXVFWAGQILM</span></span></span>
<b>testo:</b> BOB <span style="color: red;">STAI</span> ATTENTO <span style="color: green;">AEVE</span>
<b>messaggio cifrato:</b> D <span style="color: blue;">EDEFWSR</span> S <span style="color: orange;">WWBY</span> WE <span style="color: purple;">SBGB</span>

In questo modo le chiavi possibili sono ben  $26! - 1$ , cioè circa  $4 * 10^{26}$ .

**Osservazione:** Anche in questo caso, **il cifrario non è comunque sicuro**, dato che si può forzare senza ricorrere a brute-force attacks, sfruttando la struttura dei messaggi in chiaro e l'occorrenza statistica delle lettere.

### Caratteristiche generali

Più in generale, **le cifrature monoalfabetiche sono facili da violare perché conservano le informazioni di frequenza dell'alfabeto originario**, cioè la struttura del testo in chiaro permane nel testo cifrato.

#### 6.4.2 Cifrari a sostituzione polialfabetica

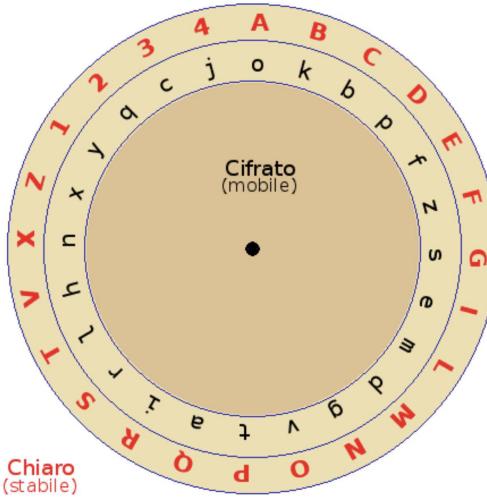
Come abbiamo detto, nei cifrari polialfabetici una stessa lettera che si incontra in punti diversi del messaggio in chiaro ammette un insieme di lettere sostitutive possibili scelte con una regola opportuna.

### Cifrario di Augusto

Esempio più antico di cifrario polialfabetico, esso funzionava nel seguente modo: Augusto scriveva i documenti in greco, poi metteva in corrispondenza la sequenza di lettere del documento con la sequenza di lettere del primo libro dell'Iliade. Quindi sostituiva ogni lettera del documento con il numero che indicava la distanza, nell'alfabeto greco, di tale lettera con quella in pari posizione nell'Iliade.

### Cifrario di Alberti

Il cifrario di Alberti basa il suo funzionamento sul disco di Leon Battista Alberti, il quale dispone di due dischi concentrici, di cui quello esterno è mobile. Il disco interno contiene caratteri disordinati, mentre quello esterno contiene caratteri ordinati ed i numeri 1, 2, 3 e 4, chiamati **nulle**.



Ci sono più metodi di cifratura definiti da Alberti. Qui ne vediamo due.

Il primo metodo è definito dai seguenti passaggi:

1. Si inseriscono delle nulle in posizioni casuali del testo in chiaro (e.g. *HELLOWORLD* → *HELL3OWOR1LD*).
2. Si definisce una chiave segreta che deve essere nota ai due che comunicano. La chiave è costituita dalla lettera del disco interno che viene posta sotto la *A* di quello esterno (e.g. *key = s*, quindi i dischi vengono ruotati affinché la *s* stia sotto la *A*).
3. Si comunica ad effettuare una decifrazione monoalfabetica finché non si incontra una nulla nel testo in chiaro. A questo punto, il carattere che corrisponde alla nulla diventerà la nuova chiave, e quindi i dischi dovranno essere ruotati affinché tale carattere finisca sotto la *A*. Si procede ciclicamente in questo modo.

#### Esempio:

A B C D E F G H I L M N O P Q R S T U V Z	1 2 3 4 5
S D T K B J O H R Z C U N Y E P X V F W A G Q I L M	

Chiave: A-S

Messaggio: NON FIDARTI DI EVE

m = **N**ON F I D A **2** R T I D I E V E  
c = **U**N U J R K S Q

A B C D E F G H I L M N O P Q R S T U V Z	1 2 3 4 5
Q I L M S D T K B J O H R Z C U N Y E P X V F W A G	

Chiave: A-Q

Messaggio: NON FIDARTI DI EVE

m = **N**ON F I D A **2** R T I D I E V E  
c = **U**N U J R K S Q U Y B M B S P S

Il secondo metodo si chiama **indice mobile** e procede nel seguente modo:

1. Si inseriscono delle nulle in posizioni casuali del testo in chiaro.
2. Si definisce una chiave segreta che deve essere nota ai due che comunicano.
3. Si comunica ad effettuare una decifrazione monoalfabetica finché non si incontra una nulla nel testo in chiaro. Il valore di tale nulla (1, 2, 3 o 4) dice tra due caratteri la chiave verrà cambiato col carattere in più che viene aggiunto nel testo in chiaro (e.g. *HELLOWORLD* → *HELL2OWORLD* → *HELL2OWTORLD*, quindi quando si incontra il due, si sa che il terzo prossimo carattere (in questo caso la *P*) sarà la nuova chiave). Si procede ciclicamente in questo modo.

**Osservazione:** *Enigma*, la macchina di cifratura usata nella seconda guerra mondiale dai tedeschi, non è altro che un'estensione elettromeccanica del cifrario di Alberti.

### Idea di Vigenèr

Ispirato al cifrario di Alberti, Vigenèr ha sviluppato un prototipo di cifrario che usa una chiave corta di lettere, in cui ogni lettera della chiave indica una traslazione della corrispondente lettera del testo. Quindi la cifratura procede col meccanismo mostrato in figura.

chiave:	C	H	I	A	V	E									
traslazione:	2	7	8	0	24	4									
	N	O	N	F	I	D	A	R	T	I	D	I	E	V	E
	2	7	8	0	24	4	2	7	8	0	24	4	2	7	8
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
	P	V	V	F	G	H	C	Y	B	I	B	M	G	C	M

**Osservazione:** Tale cifrario è costituito da una sequenza di cifrari di Cesare con shift diversi.

### Cifrario di Vigenère

Il cifrario di Vigenère si basa sull'utilizzo della seguente **tabella pubblica**:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

La chiave è costituita da una parola segreta  $k$ , e la cifratura del messaggio  $m$  avviene nel seguente modo:

1. si dispongono  $m$  e  $k$  su due righe adiacenti, allineando le lettere in verticale (se  $k$  è più corta di  $m$ , la chiave si ricopia più volte). Quindi ogni lettera  $x$  del messaggio in chiaro risulta allineata ad una lettera  $y$  della chiave.
2. la  $x$  viene sostituita nel crittogramma con la lettera che si trova nella cella all'incrocio tra la riga che inizia con  $x$  e la colonna che inizia con  $y$ .

**Esempio:** Dati  $m = ILDELFINO$  e  $k = ABRA$  si ottiene

I L	D E L F I N O
A B	R A A B R A A
↓ ↓	↓ ↓ ↓ ↓ ↓ ↓ ↓
I M	U E L G Z N O

La decifrazione del crittogramma si esegue cercando nella riga che corrisponde al carattere della chiave il corrispondente carattere del crittogramma. Il primo carattere della colonna corrisponde al carattere in chiaro.

Da un punto di vista matematico, abbiamo per la crifratura

$$pos(c_i) = (pos(m_i) + pos(k_{i \bmod h})) \bmod 26$$

e per la decifrazione

$$pos(m_i) = (pos(c_i) - pos(k_{i \bmod h})) \bmod 26$$

dove  $h$  è la lunghezza della chiave e  $pos(A) = 0$ .

**Osservazione:** La sicurezza del metodo è direttamente influenzata dalla lunghezza della chiave scelta.

### Cifrario One-Time Pad

Il cifrario One-Time Pad è un'istanza del cifrario di Vigenère, nella quale la chiave è lunga quanto il messaggio.

Caratteristica di questo cifrario è la sua inattaccabilità, cioè **non può essere decifrato senza conoscere la chiave**.

#### 6.4.3 Cifrari a trasposizione

L'idea di base è quella di eliminare qualsiasi struttura linguistica presente nel messaggio, e questo viene fatto

- permutando le lettere del messaggio in chiaro
- inserendone eventualmente altre che vengono ignorate nella decifrazione

### Cifrario a permutazione semplice

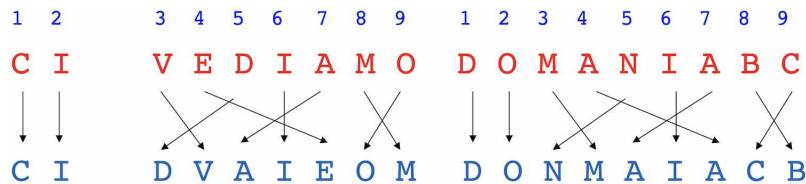
Nel cifrario a permutazione semplice, la chiave è costituita da

- un intero  $h$
- una permutazione  $\pi$  degli interi  $\{1 \dots h\}$

La cifratura avviene suddividendo il messaggio  $m$  in blocchi di  $h$  lettere, quindi si permutano le lettere in ciascun blocco secondo  $\pi$ .

**Osservazione:** Se la lunghezza di  $m$  non è divisibile per  $h$ , si aggiungono alla fine delle lettere qualsiasi (**padding**), che partecipano alla trasposizione ma sono ignorate dal destinatario perché la decifrazione le riporta alla fine del messaggio.

**Esempio:** Consideriamo  $h = 9$  e  $\pi = \{1, 2, 5, 3, 7, 6, 4, 9, 8\}$ , allora si ottiene:



**Osservazione:** Il numero delle chiavi possibili è infinito, dato che  $h$  non è prefissato. Comunque, scelto un certo  $h$ , il numero di possibili chiavi è  $h! - 1$ . Si capisce quindi che tanto maggiore è  $h$ , tanto più difficile è impostare un attacco esauriente.

### Cifrario a permutazione di colonne

Nel cifrario a permutazione di colonne la chiave è  $k = (c, r, \pi)$ , in cui

- $c$  ed  $r$  denotano il numero di colonne e di righe di una tabella di lavoro  $T$ .
- $\pi$  è una permutazione degli interi  $\{1 \dots c\}$

La cifratura avviene suddividendo il messaggio  $m$  in blocchi  $m_1, m_2, \dots$  di  $c * r$  caratteri ciascuno. Quindi, i caratteri sono distribuiti tra le celle di  $T$  in modo regolare, scrivendoli per righe dall'alto verso il basso.

$$c = 6, r = 3, \pi = \{2, 1, 5, 3, 4, 6\}$$

$m = \text{NON SONO IL COLPEVOLE}$

N	O	N	S	O	N
O	I	L	C	O	L
P	E	V	O	L	E

A questo punto le colonne di  $T$  sono permutate secondo  $\pi$ .

1 2 3 4 5 6	2 1 5 3 4 6
N O N S O N	O N O N S N
O I L C O L	I O O L C L
P E V O L E	E P L V O E
T	T permutata
$c = \boxed{O I E   N O P   O O L   N L V   S C O   N L E}$	

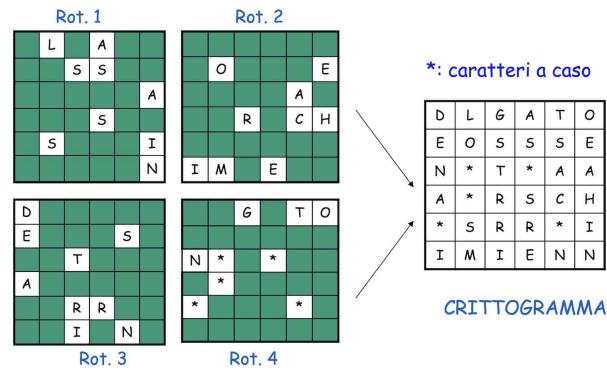
Questo lo si fa per ogni blocco.

### Cifrario a griglia

Il cifrario a griglia prevede che la chiave sia una griglia quadrata, di dimensioni  $q * q$  con  $q$  pari, in cui un quarto delle celle ( $s = q^2/4$ ) sono trasparenti, mentre le altre sono opache.

La cifratura avviene scrivendo i primi  $s$  caratteri del messaggio, nelle posizioni corrispondenti alle celle trasparenti, quindi la griglia viene rotata tre volte di 90 gradi in senso orario, e si ripete per ogni rotazione l'operazione di scrittura di tre successivi gruppi di  $s$  caratteri.

**Esempio:** Consideriamo  $q = 6$  ed il messaggio  $m = \text{"L'ASSASSINO È ARCHIMEDES TARRINGTON"}$ , allora otterremo:



**Osservazione:** La griglia deve essere scelta in modo che le posizioni corrispondenti alle celle trasparenti non si sovrappongano mai nelle quattro rotazioni. Inoltre, se la lunghezza del messaggio è minore di  $4s$ , le posizioni della pagina  $P$  rimaste vuote si riempiono con caratteri scelti a caso. Invece, se la lunghezza del messaggio è maggiore di  $4s$ , il messaggio viene decomposto in blocchi di  $4s$  caratteri ciascuno, ed ogni blocco è cifrato indipendentemente dagli altri.

**Osservazione:** La decifrazione di  $P$  è eseguita sovrapponendovi quattro volte la griglia.

**Osservazione:** Il numero di chiavi (cioè griglie) possibili è  $4^s$ . Quindi basta avere, per esempio,  $q = 12$  per stare sicuri contro brute-force attacks.

# Chapter 7

## Lezione 7 (11/10/2022)

Come sappiamo, la sicurezza di un cifrario è legata alla dimensione dello spazio delle chiavi. Infatti, un piccolo spazio delle chiavi rende agevole attacchi di tipo brute-force.

Tuttavia, esistono molte altre tipologie di attacco, cosicché avere **un grande spazio delle chiavi non basta a rendere il cifrario sicuro**. Per esempio, tutti i cifrari storici sono stati infatti violati con un attacco statistico di tipo **cipher text**, in cui il crittoanalista ha a disposizione solo il crittogramma e qualche informazione sull'ambiente in cui esso è stato generato.

### 7.1 Crittoanalisi statistica

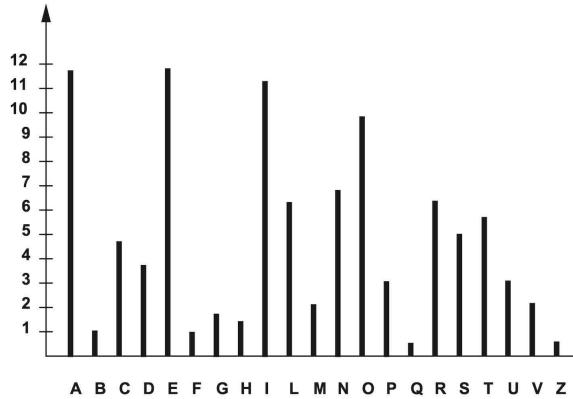
In generale si fanno le seguenti assunzioni:

- il crittoanalista conosce il metodo impiegato per la cifratura/decifrazione
- il crittoanalista conosce il linguaggio naturale nel quale è scritto il messaggio in chiaro
- il messaggio in chiaro è sufficientemente lungo per poter rilevare alcuni dati statistici sui caratteri che compongono il crittogramma

Queste sono tutte assunzioni più che lecite. Infatti, è quasi certo che prima o poi il metodo di cifratura venga scoperto. Inoltre, il crittoanalista spesso sa chi vuole spiare, sapendo quindi anche la lingua utilizzata nei messaggi. Infine, la gran parte dei messaggi è abbastanza lunga da soddisfare la terza assunzione.

L'attacco di base sull'utilizzo di un insieme di **tavole note**. È infatti ben studiata in ogni lingua la frequenza con cui appaiono in media le varie lettere dell'alfabeto.

**Esempio:** Di seguito è rappresentata la frequenza in percentuale delle varie lettere nell'italiano:



**Osservazione:** Dati simili sono noti per le frequenze di digrammi, trigrammi, etc.

### 7.1.1 Crittogrammi per sostituzione monoalfabetica

Se un crittogramma è stato generato per sostituzione monoalfabetica, la frequenza con cui vi appare una lettera  $y$ , corrispondente alla  $x$  del messaggio, sarà ragionevolmente uguale alla frequenza della  $x$  in italiano. Ovviamente, questa stima è tanto più accuarata quanto più il messaggio è lungo.

Confrontando le frequenze delle lettere come appaiono nel crittogramma con quelle dell'italiano, e provando alcune permutazioni tra lettere di frequenze simili, si ottengono diverse ipotesi di prima decifrazione.

**Esempi:**

- Nel cifrario di Cesare, basta scoprire la corrispondenza di una sola coppia  $(y, x)$  per trovare la chiave e descrittare il messaggio.
- Nei cifrari affini, basta individuarne due di coppie per trovare i parametri  $a$  e  $b$  che formano la chiave segreta.
- Anche la decifrazione in un cifrario completo, ove la chiave ‘è una permutazione arbitraria dell’alfabeto, non presenta difficolta’ se si associano le lettere in base alle frequenze.

### 7.1.2 Crittogrammi per sostituzione polialfabetica

Assai più difficile è la decifrazione nei cifrari a sostituzione polialfabetica, dato che ciascuna lettera  $y$  del crittogramma dipende da una coppia di lettere  $(x, z)$  provenienti dal messaggio e dalla chiave. Questo rende quasi inutile guardare alla frequenza delle lettere nel crittogramma.

Questi cifrari hanno tuttavia un punto di debolezza se la chiave è unica e ripetuta più volte per coprire l’intero messaggio (come nel cifrario di Vigenère). Infatti, se la chiave contiene  $h$  caratteri, allora il crittogramma può essere decomposto in  $h$  sottosequenze, ciascuna delle quali è stata ottenuta per sostituzione monoalfabetica.

### 7.1.3 Crittogrammi per trasposizione

Non ha senso condurre un attacco statistico sui crittogrammi ottenuti per trasposizione, dato che le lettere che appaiono nel crittogramma sono le stesse del messaggio in chiaro. Tuttavia di possono

sfruttare altre tecniche che si poggiano sullo studio dei **q-grammi**.

Consideriamo, per esempio, un cifrario a permutazione semplice. Se si conosce la lunghezza  $h$  della chiave, il crittogramma si divide in porzioni di pari lunghezza e in ciascuna di esse si cercano i gruppi di  $q$  lettere (anche non adiacenti) che formano i q-grammi più comuni del linguaggio del messaggio. Se uno di questi gruppi deriva effettivamente da un q-gramma del messaggio, allora viene scoperta la parte di permutazione che ha permuto l'ordine delle lettere di quel q-gramma.

## 7.2 Enigma

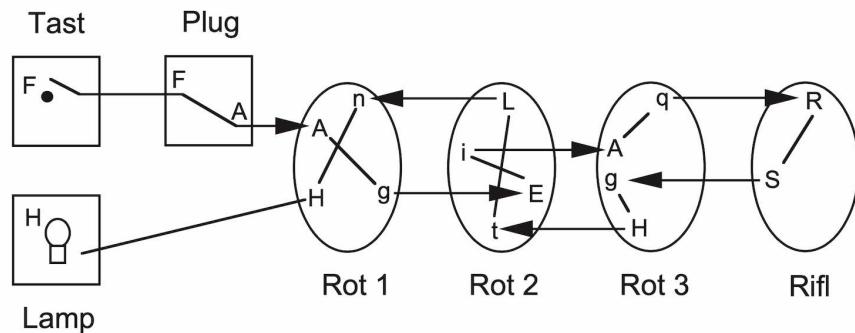
Parliamo adesso di Enigma, il famoso sistema di cifratura utilizzato dai tedeschi nella seconda guerra mondiale. Esso è nato come estensione elettromagnetica del cifrario di Alberti.

### 7.2.1 Modo d'uso

Il sistema si presentava all'esterno come una cassetta dotata di una tastiera simile a quella delle macchine da scrivere del tempo, con ventisei tasti corrispondenti alle lettere dell'alfabeto tedesco; e di ventisei lampadine corrispondenti anch'esse alle lettere. Sulla tastiera si batteva il messaggio e le lampade indicavano il crittogramma che andava creandosi carattere per carattere; questo era tipicamente copiato con carta e penna e poi spedito al destinatario attraverso altri mezzi (in genere radio o telegrafo). Poiché il sistema aveva una struttura perfettamente reversibile, il ricevente batteva sulla tastiera il crittogramma per ricostruire il messaggio sulle lampadine.

### 7.2.2 Struttura interna

Il cuore di Enigma era costituito da tre dischi di gomma rigida (i **rotori**) montati sullo stesso asse, ma liberi di rotare in modo indipendente, e da un disco fisso (il **riflettore**). Ciascun disco era connesso al successivo attraverso dei contatti elettrici. La tastiera era collegata al primo rotore, ma durante lo sviluppo del sistema, tra questi due elementi fu inserito un pannello di controllo (il **plugboard**). Le due facce di ogni rotore, dette rispettivamente di **pad** e di **pin**, erano dotate di 26 contatti elettrici disposti in cerchio, corrispondenti ai caratteri dell'alfabeto, ed il rotore conteneva al suo interno un cablaggio fisso che collegava a due a due i contatti di una faccia con quelli dell'altra: il rotore conteneva quindi, tra pad e pin, una permutazione fissa delle lettere. Ogni tasto della tastiera era collegato elettricamente (tramite plugboard) a un contatto del pad del primo rotore; ogni contatto pin di un rotore toccava il contatto pad del successivo rotore disposto nelle stessa posizione rispetto all'asse; i contatti pin del terzo rotore toccavano i contatti pad del riflettore, che aveva solo contatti di questo tipo cablati a due a due tra di loro.



### 7.2.3 Peculiarità

Una caratteristica fondamentale era che i rotori non mantenevano mai la stessa posizione reciproca durante la cifratura. Infatti, per ogni lettera battuta dall'operatore, il primo rotore avanzava di un passo; dopo 26 passi il rotore era tornato nella posizione iniziale ed avanzava di un passo il secondo rotore; quando anche questo aveva fatto una rotazione completa avanzava di un passo il terzo rotore.

A causa dello spostamento dei rotori la corrispondenza tra caratteri cambiava ad ogni passo.

### 7.2.4 Sicurezza del sistema

Ogni rotore realizza una permutazione fissa dell'alfabeto. Ciò implica che se i rotori fossero fermi, allora le tre permutazioni si comporrebbero tra loro in un'unica permutazione risultante, ottenendo quindi una sola chiave monoalfabetica la cui sicurezza è praticamente nulla.

Poiché però i rotori si muovono uno rispetto all'altro, **la chiave cambia a ogni passo**, all'interno di  $26^3 = 17576$  chiavi possibili. Tuttavia, siccome i rotori sono oggetti fisici immutabili, quelle chiavi sarebbero sempre le stesse, applicate sempre nello stesso ordine. Questo renderebbe il sistema facilmente forzabile. Inoltre, chiunque avesse posseduto una macchina Enigma, avrebbe potuto immediatamente decifrare il messaggio!

Per rendere il sistema sicuro, vennero attuati vari meccanismi.

#### Rotori permutanti tra loro

In primis, la macchina fu costruita in modo tale che **i tre rotori potessero essere permutati tra loro**, così da arrivare ad avere  $17576 * 6 = 105456$  chiavi possibili.

#### Plugboard

Inoltre, il plugboard consentiva di scambiare tra loro i caratteri di sei coppie scelte **arbitrariamente** in ogni trasmissione mediante fili muniti di spinotti. In tal modo, considerando che ogni cablaggio può essere descritto come una sequenza di 12 caratteri interpretati a due a due come le coppie da scambiare, si potevano realizzare  $\binom{26}{12} > 10^{10}$  combinazioni imprevedibili.

In complesso l'introduzione del plugboard ha portato le chiavi possibili a più di  $10^{20}$ .

#### Ulteriori modifiche

Inoltre durante la seconda guerra mondiale i tedeschi dotarono le macchine Enigma di otto rotori diversi di cui ne venivano montati di volta in volta tre, stabilirono che le posizioni iniziali mutue dei rotori fossero scelte arbitrariamente, ed aumentarono a dieci le coppie di caratteri scambiabili nel plugboard.

### 7.2.5 Decifrazione

Ogni trasmissione iniziava con una parte, a sua volta cifrata, che descriveva l'assetto complessivo dei rotori e del plugboard. Più precisamente, i reparti militari possedevano un segretissimo elenco di **chiavi giornaliere** ciascuna delle quali stabiliva l'assetto iniziale della macchina per quel giorno. Con questo assetto si trasmetteva una nuova chiave di messaggio che indicava il nuovo assetto da usare in quella particolare trasmissione.

# Chapter 8

## Lezione 8 (17/10/2022)

### 8.1 Cifrari perfetti

Nel 1949, Claude Shannon ha formalizzato matematicamente il concetto di **cifrario perfetto**, cioè di cfrario incodnizionatamente sicuro. Inviolabile, insomma.

Nella comunicazione tra due utenti, il comportamento del mittente è descritto da una variabile aleatoria **M** che assume valori nello spazio **MSG** dei messaggi, e le comunicazioni sul canale sono descritte da una variabile aleatoria **C** che assume valori nello spazio **CRI** dei crittogrammi. Quindi definiamo:

- $P(\mathbf{M} = m)$ , la probabilità che il mittente voglia spedire il messaggio  $m$  al destinatario.
- $P(\mathbf{M} = m | \mathbf{C} = c)$ , la probabilità che, dato il crittogramma  $c$ , il messaggio in chiaro sia  $m$ .

**Definizione:** Un cfrario è perfetto se

$$\forall m \in \mathbf{MSG} \wedge c \in \mathbf{CRI} . P(\mathbf{M} = m | \mathbf{C} = c) = P(\mathbf{M} = m)$$

In altre parole, un cfrario è perfetto se **nessuna** informazione sul messaggio originale può essere ricavata dalla conoscenza del relativo crittogramma, in assenza della chiave di decifrazione.

Impiegando un cfrario perfetto, la conoscenza complessiva del crittoanalista non cambia dopo che egli ha osservato un crittogramma arbitrario  $c$  transitare sul canale.

**Teorema di Shannon:** *In un cfrario perfetto il numero delle chiavi deve essere maggiore o uguale al numero dei messaggi possibili.*

#### Dimostrazione

Sia  $N_m$  il numero dei messaggi possibili, cioè con  $P(\mathbf{M} = m) > 0$ , e  $N_k$  il numero delle chiavi. Poniamo per assurdo  $N_m > N_k$ . Ad un crittogramma  $c$ , con  $P(\mathbf{C} = c) > 0$ , corrispondo  $s \leq N_k$  messaggi ottenuti decrittando  $c$  con tutte le chiavi. Poiché  $N_m > N_k \geq s$ , esiste almeno un messaggio  $m$ , con  $P(\mathbf{M} = m) > 0$ , non ottenibile da  $c$ . Questo implica che  $P(\mathbf{M} = m | \mathbf{C} = c) = 0$ , concludendo che il cfrario non è perfetto.

**Osservazione:** Il teorema di Shannon dimostra che l'uso di cfrari perfetti è necessariamente molto costoso, poiché richiede chiavi lunghissime per descrivere l'intero spazio delle chiavi. I cfrari perfetti sono dunque inadatti a una crittografia di massa, ma risultano estremamente attraenti per chi richieda una sicurezza assoluta e sia disposto a pagarne i costi.

### 8.1.1 Cifrario One-Time Pad

Il cifrario One-Time Pad (**OTP**), è il più famoso e utilizzato dei i cfrari perfetti.

Le funzioni di cifratura e decifrazione eseguono trasformazioni tra sequenze di bit preservando la loro lunghezza: dato un messaggio  $m$  di  $n$  bit e una chiave  $k$ , la cifratura produrrà un crittogramma  $c$  anch'esso di  $n$  bit. Tali trasformazioni sono basate sullo **XOR** ( $\oplus$ ).

#### Generazione della chiave segreta

Si costruisce una sequenza  $k = k_1 k_2 \dots$  di bit, nota sia al mittente che al destinatario, avente lunghezza maggiore o uguale a quella del messaggio da scambiare. Ogni bit  $k_i$  deve essere scelto **perfettamente a caso**.

#### Cifratura

Dato il messaggio di  $n$  bit  $m = m_1 m_2 \dots m_n$ , il crittogramma  $c = c_1 c_2 \dots c_n$  viene generato bit a bit ponendo  $c_i = m_i \oplus k_i$ .

#### Decifrazione

Il messaggio  $m$  è ricostruito bit a bit ponendo  $m_i = c_i \oplus k_i$  (infatti  $a \oplus b \oplus b = a$ ).

**Osservazione:** Un problema molto serio è quello della disponibilità di una chiave arbitrariamente lunga e perfettamente casuale.

**Teorema:** *Supponendo che*

- tutti i messaggi abbiano la stessa lunghezza  $n$ ,
- tutte le sequenze di  $n$  bit siano messaggi possibili,
- si utilizzi una chiave scelta perfettamente a caso per ogni messaggio,

*il cifrario One-Time Pad è perfetto ed impiega un numero minimo di chiavi.*

#### Dimostrazione

Per definizione di probabilità condizionata, vale che

$$P(\mathbf{M} = m | \mathbf{C} = c) = P(\mathbf{M} = m \wedge \mathbf{C} = c) / P(\mathbf{C} = c)$$

Adesso, siccome ogni chiave viene generata con probabilità  $(\frac{1}{2})^n$  e chiavi diverse danno origine a crittogrammi diversi (per le proprietà dello XOR), si deduce che  $P(\mathbf{C} = c) = (\frac{1}{2})^n$ . Quindi, ogni crittogramma è equiprobabile e conseguentemente indipendente da  $m$ . Questo implica che

$$P(\mathbf{M} = m \wedge \mathbf{C} = c) = P(\mathbf{M} = m) * P(\mathbf{C} = c)$$

e quindi

$$P(\mathbf{M} = m | \mathbf{C} = c) = P(\mathbf{M} = m \wedge \mathbf{C} = c) / P(\mathbf{C} = c) = P(\mathbf{M} = m) * P(\mathbf{C} = c) / P(\mathbf{C} = c) = P(\mathbf{M} = m)$$

Si conclude perciò che il cifrario One-Time Pad è perfetto.

# **Chapter 9**

## **Lezione 9 (17/10/2022)**

Sono stati fatti solo esercizi. Quando li rifarò li metterò qui.

# Chapter 10

## Lezione 10 (18/10/2022)

### 10.1 Cifrari simmetrici

I criteri alla base dei cifrari simmetrici sono due: **diffusione** e **confusione**.

**Osservazione:** Essi furono proposti da Shannon.

#### Diffusione

Il criterio della **diffusione** consiste nell'alterare la struttura del testo in chiaro *spargendone* i caratteri su tutto il testo cifrato.

Ciò si può ottenere **permutando** i caratteri (o i bit) del messaggio come avviene nei cifrari a trasposizione, oppure **combinando** tra loro i caratteri del messaggio in modo che ciascun carattere del crittogramma venga a dipendere da molti di essi.

**Osservazione:** Col primo modo, si fa perdere l'informazione sulla frequenza dei q-grammi, col secondo si perde anche la frequenza delle singole lettere.

#### Confusione

Il criterio della **confusione** consiste invece nel combinare in modo complesso il messaggio e la chiave per non permettere al crittoanalista di separare queste due sequenze mediante un'analisi del crittogramma.

**Osservazione:** Ciò si ottiene nei cifrari a sostituzione polialfabetica in misura tanto maggiore quanto più lunga è la chiave, e in modo virtualmente perfetto nel cifrario One-Time Pad.

### 10.2 DES

Il DES (Data Encryption Standard) è un **cifrario simmetrico** di uso generale con la seguente struttura:

- Il messaggio è suddiviso in **blocchi di 64 bit**, ciascuno dei quali viene cifrato e decifrato indipendentemente dagli altri.

- Cifratura e decifrazione procedono attraverso **16 fasi** successive (**round**) in cui si ripetono le stesse operazioni.
- La chiave segreta  $k$  è composta da **8 byte** (64 bit). In ciascun byte, i primi 7 bit sono scelti arbitrariamente, mentre l'ottavo è aggiunto per il controllo di parità.
- Dala chiave  $k$  vengono create **16 sottochiavi**  $k[0], k[1], \dots$  impiegate una per fase.
- Il messaggio viene diviso in due metà  $S$  e  $D$  (sinistra e destra). In ciascuna fase, si eseguono le operazioni

$$S = D \text{ e } D = f(k[i-1], D, S)$$

in cui  $f$  è un'opportuna funzione **non lineare** ed  $i = 1, \dots, 16$ .

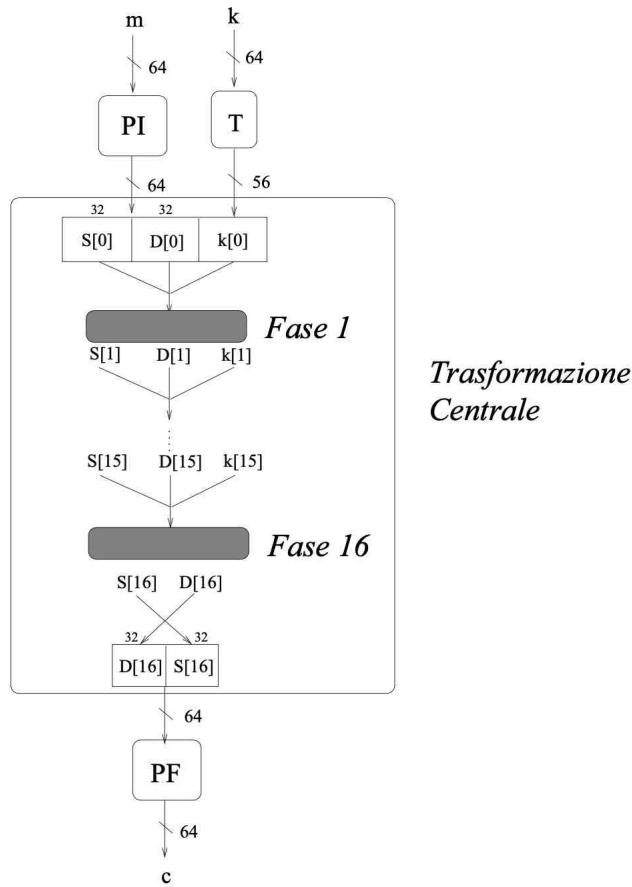
Alla fine delle 16 fasi, le due metà vengono nuovamente scambiate e poi concatenate per produrre il crittogramma finale.

- La decifrazione consiste nel ripetere il processo invertendo l'ordine delle chiavi.

**Osservazione:** L'insieme delle operazioni coinvolte nella cifratura garantisce che, alla fine del processo, ogni bit del crittogramma dipenda da tutti i bit della chiave e da tutti i bit del messaggio in chiaro, rispondendo così ai criteri di confusione e diffusione.

### 10.2.1 Cifratura generale

In prima approssimazione, le operazioni del DES sono schematizzate in figura:



in cui  $m$  è un blocco del messaggio,  $c$  è il corrispondente blocco del crittogramma,  $k$  è la chiave segreta.

### Permutazioni iniziali

- $PI$  permuta i 64 bit del messaggio in chiaro, mentre  $T$  depura la chiave dai bit di parità e permuta i rimanenti 56 bit per generare la prima sottochiave  $k[0]$ .

### Trasformazione centrale

- La sequenza permutata da  $PI$  viene decomposta in un blocco sinistro  $S[0]$  ed un blocco destro  $D[0]$  di 32 bit ciascuno.
- Su questi due blocchi si eseguono, nelle successive fasi del DES, 16 trasformazioni strutturalmente uguali, ciascuna con una diversa sottochiave  $k[i]$  derivata dalla chiave iniziale  $k$ .
- La  $i$ -esima fase riceve in ingresso tre blocchi  $S[i - 1], D[i - 1], k[i - 1]$  e produce in uscita tre nuovi blocchi  $S[i], D[i], k[i]$ .
- Dopo l'ultima fase, i due blocchi  $S[16]$  e  $D[16]$  vengono scambiati e concatenati tra loro in un unico blocco di 64 bit, dal quale viene costruito il crittogramma finale.

### Permutazione finale

- $PF$  genera la permutazione inversa di  $PI$ , rimescolando nuovamente i bit del crittogramma.

58 50 42 34 26 18 10 2
60 52 44 36 28 20 12 4
62 54 46 38 30 22 14 6
64 56 48 40 32 24 16 8
57 49 41 33 25 17 9 1
59 51 43 35 27 19 11 3
61 53 45 37 29 21 13 5
63 55 47 39 31 23 15 7

Permutazione  $PI$ 

40 8 48 16 56 24 64 32
39 7 47 15 55 23 63 31
38 6 46 14 54 22 62 30
37 5 45 13 53 21 61 29
36 4 44 12 52 20 60 28
35 3 43 11 51 19 59 27
34 2 42 10 50 18 58 26
33 1 41 9 49 17 57 25

Permutazione  $PF$ 

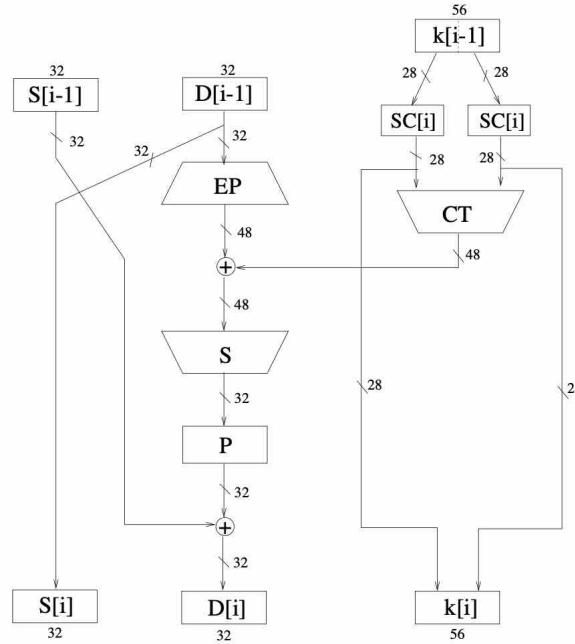
57 49 41 33 25 17 9
1 58 50 42 34 26 18
10 2 59 51 43 35 27
19 11 3 60 52 44 36
63 55 47 39 31 23 15
7 52 54 46 38 30 22
14 6 61 53 45 37 29
21 13 5 28 20 12 4

Trasposizione  $T$ 

Da notare come  $PF$  sia la permutazione inversa di  $PI$ , e come  $T$  scarti dalla chiave gli 8 bit di parità.

#### 10.2.2 Singola fase

Avendo visto in generale come avviene la cifratura del messaggio, studiamo adesso in maggior dettaglio le singole fasi del DES. Vediamo in realtà solo la fase  $i$ -esima, dato che sono tutte strutturalmente identiche.



### Shift ciclico

- La sottochiave  $k[i - 1]$  di 56 bit, ricevuta dalla fase precedente, viene suddivisa in due metà di 28 bit ciascuna. Su ognuna di queste, la funzione  $SC[i]$  esegue uno **shift ciclico verso sinistra** di un numero di posizioni definito come segue:

$$SC[i] = \begin{cases} 1 & i = 1, 2, 9, 16, \\ 2 & \text{altrimenti} \end{cases}$$

- Le due parti così traslate vengono concatenate in un unico blocco di 56 bit, il quale costituisce la sottochiave  $k[i]$  per la fase successiva.

### Compressione e trasposizione

- Il blocco di 56 bit prodotto dall'operazione precedente viene ulteriormente elaborato per produrre un nuovo blocco di 48 bit, utilizzato nel processo di cifratura dei blocchi  $S[i-1]$  e  $D[i-1]$ .

La funzione  $CT$  esegue una permutazione del blocco e una selezione di 48 bit da esso nel seguente modo:

14	17	11	24	01	05
03	28	15	06	21	10
23	19	12	04	26	08
16	07	27	20	13	02
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

**Osservazione:** La combinazione delle funzioni  $SC[i]$  e  $CT$ , garantisce che in ogni fase venga estratto dalla sottochiave un diverso sottoinsieme di bit per la cifratura.

**Osservazione:** Si calcola che nella cifratura ogni bit della chiave originale  $k$  partecipi in media a quattordici fasi.

### Espansione e permutazione

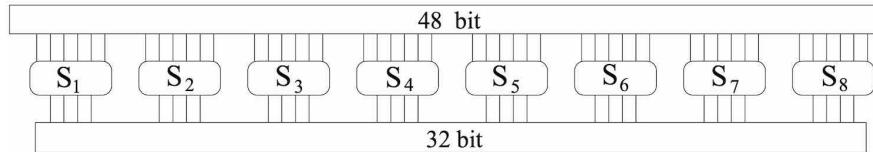
- Il nuovo  $S[i]$  è semplicemente uguale a  $D[i - 1]$ .
- Il blocco  $D[i - 1]$  di 32 bit generato prima, viene espanso a 48 bit, duplicando 16 bit in ingresso e spostandone altri nel seguente modo:

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

al fine di ottenere un blocco della stessa dimensione di quello estratto dalla sottochiave e poter eseguire lo XOR tra i due.

### Sostituzione

- La funzione  $S$  ( **$S$ -box**) consta di 8 funzioni combinatorie  $S_1, S_2, \dots$ . L'ingresso di 48 bit viene decomposto in 8 blocchi  $B_1, B_2, \dots$  di 6 bit ciascuno, i quali costituiscono l'ingresso delle sottofunzioni stesso indice.



$x \backslash y$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S_1$ 0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Sia  $B_j = b_1 b_2 \dots b_6$ . Questi bit vengono divisi nei gruppi  $b_1 b_6$  e  $b_2 b_3 b_4 b_5$ , utilizzati per accedere alla cella di riga  $x$  e colonna  $y$  in una tabella che definisce la sottofunzione  $S_j$ . Il numero ivi contenuto è compreso tra 0 e 15, ed è quindi rappresentato con 4 bit, i quali costituiscono l'uscita di  $S_j$ , realizzando una compressione da 6 a 4 bit. Complessivamente, gli otto blocchi generano una sequenza di 32 bit.

**Osservazione:** I blocchi  $EP$  ed  $S$  sono studiati in modo tale che tutti i bit di  $D[i - 1]$  influenzino l'uscita di  $S$ .

### Permutazione

- La funzione  $P$  è una permutazione di 32 bit che genera il blocco finale  $D[i]$ . Essa permuta nel seguente modo:

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

- Il nuovo  $D[i]$  viene ottenuto eseguendo lo XOR tra i 32 bit della permutazione e  $S[i - 1]$ .

Tutte le funzioni applicate dal cifrario, **ad eccezione della S-box**, sono lineari rispetto allo XOR, cioè

$$f(x) \oplus f(y) = f(x \oplus y)$$

ove  $f$  è una permutazione, espansione o compressione di un vettore binario, e  $x, y$  sono vettori binari arbitrari.

**Osservazione:** La sicurezza del cifrario è garantita proprio dal fatto che le funzioni della S-box sono **non lineari** rispetto allo XOR.

### 10.3 Attacchi al DES

Poiché la chiave è formata da 56 bit, si dovrebbero idealmente provare tutte le  $2^{56}$  chiavi possibili per condurre un brute-force attack al sistema. Tuttavia, alcune osservazioni sulla struttura del cifrario permettono di ridurre leggermente lo spazio da esplorare.

Denotiamo con  $\mathbf{C}_{DES}$  e  $\mathbf{D}_{DES}$  le funzioni complessive di cifratura e decifrazione.

Si possono anzitutto escludere sessantaquattro chiavi deboli, non utilizzabili perché compromettono da sole la sicurezza del cifrario.

Una riduzione più interessante è lagata al seguente fatto:

$$\mathbf{C}_{DES}(m, k) = c \implies \mathbf{C}_{DES}(\bar{m}, \bar{k}) = \bar{c}$$

in cui la barra indica la complementazione bit a bit.

Basandosi su questa proprietà, si può condurre un *chosen plain-text* attack nel seguente modo:

- È sufficiente che il crittoanalista si procuri alcune coppie messaggio/crittogramma del tipo  $(m, c_1)$  e  $(\bar{m}, c_2)$ .
- Partendo da una delle due coppie, si sceglie una chiave  $k$  per calcolare  $\mathbf{C}_{DES}(m, k)$ .
- Se  $\mathbf{C}_{DES}(m, k) = c_1$ , allora  $k$  è probabilmente la chiave segreta (non è sicuro, dato che più chiavi portano allo stesso crittogramma).
  - Si prova allora  $k$  su altre coppie messaggio/crittogramma. Se il successo si ripete per alcune volte consecutive, allora si ha praticamente la certezza di aver trovato la chiave.

- Se invece  $\mathbf{C}_{DES}(m, k) = \bar{c}_2$ , allora  $\bar{k}$  è probabilmente la chiave segreta (in quanto genera  $(\bar{m}, c_2)$ ).
  - Come nel caso precedente, si ripete la prova su altre coppie.
- Se i due casi precedenti falliscono, si procede provando un'altra chiave.

**Osservazione:** In questo modo, le chiavi possibili si riducono a  $2^{55}$ , la metà, visto che, provata una chiave, non è necessario ripetere l'operazione sul suo complemento.

### 10.3.1 Altri attacchi

Esistono altri attacchi, tra cui:

- **Crittoanalisi differenziale** (1990) di tipo chosen plain-text
- **Crittoanalisi lienare** (1993) di tipo chosen plain-text
- Nel 2008 è stata costruita la macchina RIVYERA: un calcolatore costruito appositamente per rompere il cifrario in meno di ventiquattr'ore.

## 10.4 Variazioni del DES

Le variazioni del DES hanno sostanzialmente lo scopo di preservare la semplicità del cifrario, espandendone allo stesso tempo lo spazio delle chiavi in modo da rendere improponibili gli attacchi precedenti.

I metodi più interessanti sono la *scelta indipendente delle sottochiavi* e la *cifratura multipla*.

### 10.4.1 Scelta indipendente della sottochiavi

In questa variazione del DES, le sedici sottochiavi di 48 bit utilizzate nelle successive fasi vengono scelte **indipendentemente l'una dall'altra**, per un totale di 768 bit anziché 56. Ciò rende impraticabili i brute-force attacks.

Il problema è tuttavia il grande numero di bit che un utente deve generare per costruire la chiave.

### 10.4.2 Cifratura multipla

Più rilevante nella pratica è la cifratura multipla, la quale prevede la **concatenazione di più copie del DES** che utilizzano chiavi diverse. Si costruisce in questo modo un **cifrario composto**, che però non garantisce in linea di principio un aumento della sicurezza del cifrario complessivo rispetto ai componenti. Tuttavia, **nel caso del DES la concatenazione di più copie del cifrario non è equivalente ad una singola applicazione del cifrario stesso**.

#### 2/3-TDEA

La più importante applicazione della cifratura multipla prende vita con il **TDEA** (**Triple data Encryption Algorithm**) a 2 o 3 chiavi (rispettivamente **2TDEA** o **3TDEA**).

La cifratura col **2TDEA** è realizzata come

$$c = \mathbf{C}_{DES}(\mathbf{D}_{DES}(\mathbf{C}_{DES}(m, k_1), k_2), k_1)$$

ove  $k_1, k_2$  sono chiavi indipendenti di 56 bit segreti concordate dai due utenti. Quindi la chiave segreta risulta essere di 129 bit, di cui 112 segreti e 16 di parità.

La decifrazione è realizzata come

$$m = \mathbf{D}_{DES}(\mathbf{C}_{DES}(\mathbf{D}_{DES}(c, k_1), k_2), k_1)$$

**Osservazione:** Scegliendo  $k_1 = k_2$ , il cifrario diviene equivalente a un DES singolo.

**3TDEA** incrementa la sicurezza del metodo utilizzando tre chiavi distinte anziché due.

### Altri cifrari

Esisotono molti altri cifrari ispirati al DES. Due esempi sono **RC5** e **IDEA**, che qui non spieghiamo. Per una breve introduzione ad essi, guardare il libro di testo.

# Chapter 11

## Lezione 11 (24/10/2022)

Negli anni '90, il DES era ormai considerato non sufficientemente sicuro per le comunicazioni non classificate. Così, nell'ottobre del 2000, fu rimpiazzato dall'AES.

### 11.1 AES

L'AES (**Advanced Encryption Standard**) è costituito dall'algoritmo di cifratura *Rijndael*, proposto dalla *Proton World International*.

#### 11.1.1 Macrostruttura

L'AES è un **cifrario a blocchi**, con blocchi di 128 bit e chiavi di 128, 192 o 256 bit.

Analogamente al DES, anche l'AES prevede dei processi di criptazione a decifrazione **per fasi**. In particolare:

- 128 bit key  $\Rightarrow$  10 fasi
- 192 bit key  $\Rightarrow$  12 fasi
- 256 bit key  $\Rightarrow$  14 fasi

Prevedibilmente, ogni fase impiega una propria **chiave locale**, creata a partire dalla chiave segreta del ciphario mediante un opportuno processo di **espansione e selezione**. Un'ulteriore **chiave iniziale** è impiegata all'inizio delle operazioni.

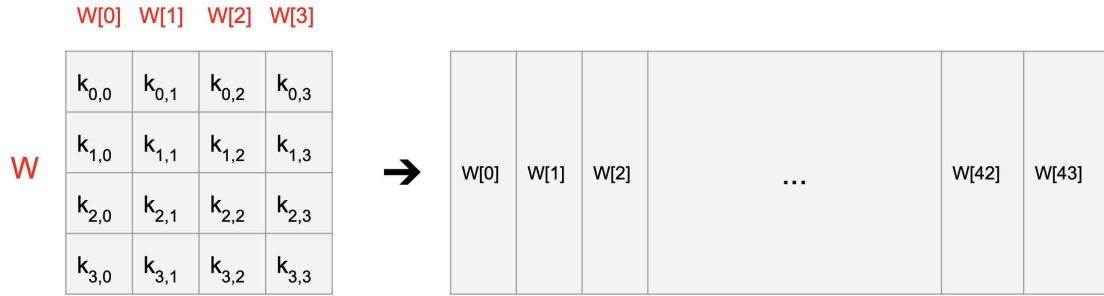
**Nota:** Descriveremo il funzionamento solo del ciphario con **chiave di 128 bit**.

#### 11.1.2 Gestione delle chiavi

La chiave iniziale è caricata **per colonne** in una matrice  $W$  di 16 byte.

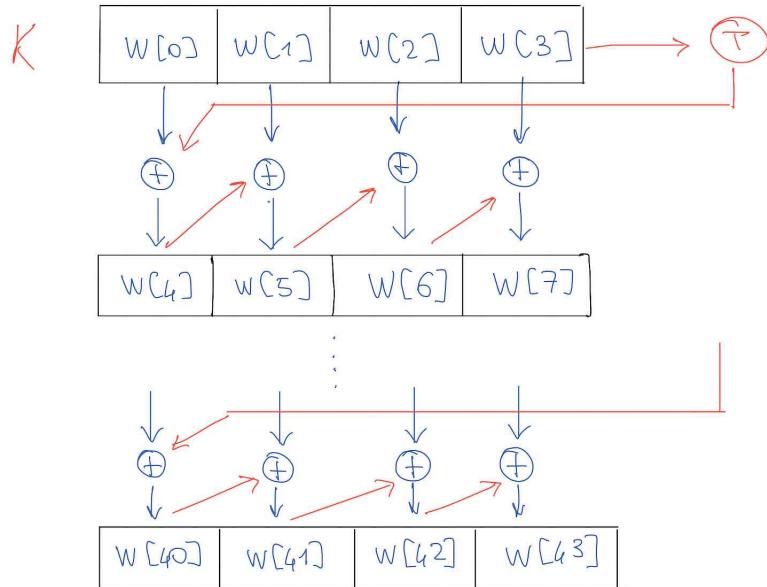
$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

A questo punto,  $W$  viene **ampliata** aggiungendo 40 colonne, generate ricorsivamente a partire dalle prime quattro.



in cui

$$W[i] = \begin{cases} W[i-4] \oplus T(W[i-1]) & i \text{ multiplo di } 4 \\ W[i-4] \oplus W[i-1] & \text{altrimenti} \end{cases}$$

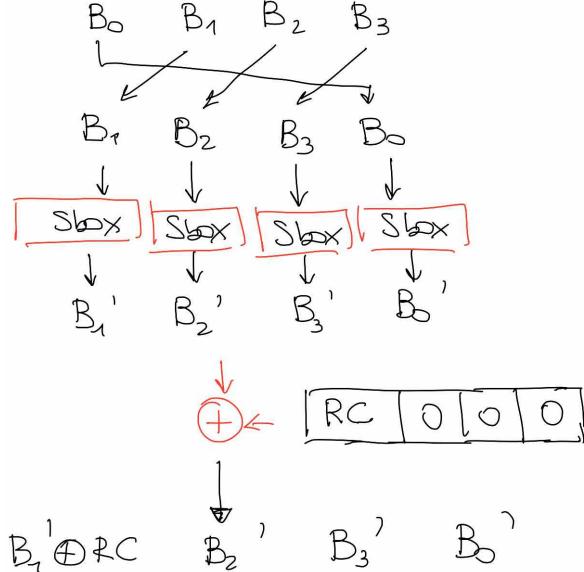


ove  $T$  è una funzione non lineare divisa in tre passi:

- **shift ciclico** verso sinistra dei 4 byte  $B_0B_1B_2B_3$  di  $W[i-1]$ ;
- applicazione della ***S-box*** su ciascun byte;

- XOR del byte più a sinistra ( $B_1'$ ) con una costante di *round RC*, differente per ogni round. In particolare, abbiamo che la *RC j-esima* ( $RC_j$ ) è calcolata come:

$$RC[j] = \begin{cases} 1 & j = 1 \\ 2RC_{j-1} & j > 1 \text{ (moltiplicazione su } GF(2^8)) \end{cases}$$



La chiave  $k(i)$  per la fase  $i$ -esima è data dalle 4 colonne  $W[4i], W[4i+1], W[4i+2], W[4i+3]$ .

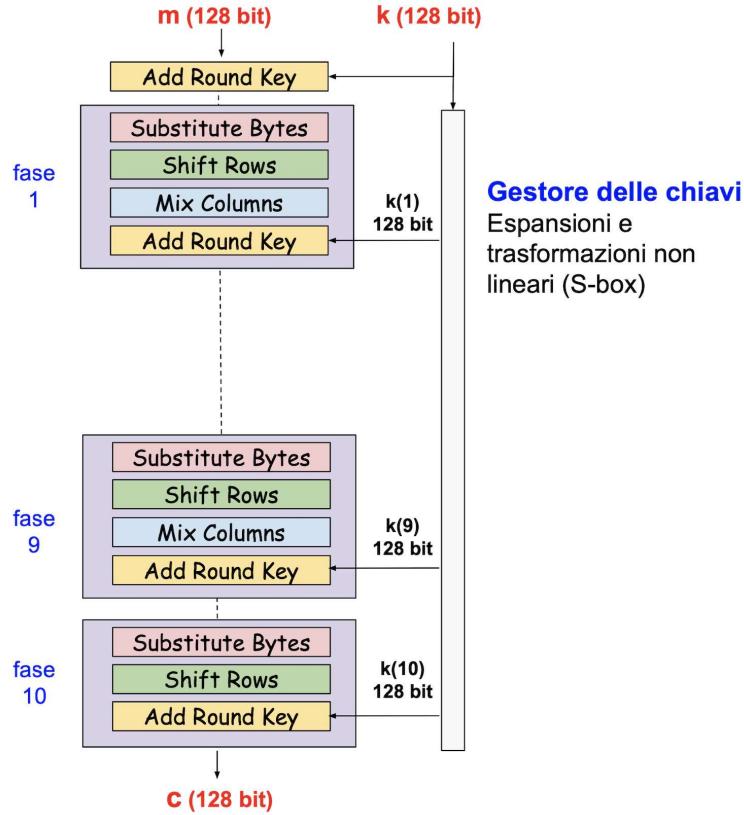
**Osservazione:** La *S-box* applicata nella funzione  $T$  è simile ma non uguale a quella del DES. Prevedibilmente, è stata resa più sicura.

### 11.1.3 Processo di cifratura

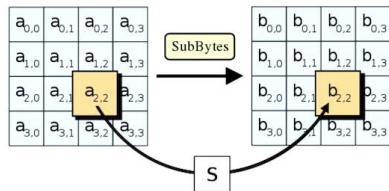
Ogni blocco è logicamente organizzato per righe come una matrice bidimensionale  $B$  di 16 byte.

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$
$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$
$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$
$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$

Ogni blocco viene quindi cifrato singolarmente. Lo schema generale è il seguente:

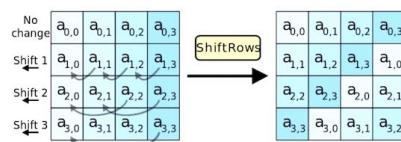


Come si può osservare, ogni fase effettua 4 operazioni:



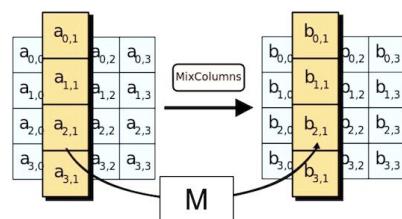
### 1. Substitution byte

ogni byte del blocco B è trasformato mediante una **S-box**: una look-up table che contiene una permutazione di tutti i 256 interi a 8 bit



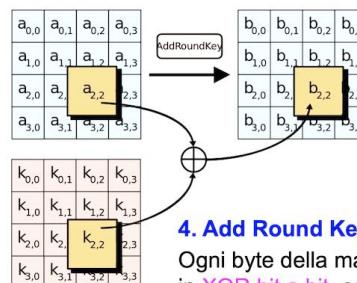
### 2. Shift Rows

I byte di ogni riga vengono shiftati verso sinistra di 0, 1, 2 e 3 posizioni, rispettivamente → i 4 byte di ogni colonna si disperdono su 4 colonne diverse



### 3. Mix Columns

$a_{ij}$  si trasforma linearmente in un nuovo byte  $b_{ij}$  che dipende da tutti i byte della colonna j

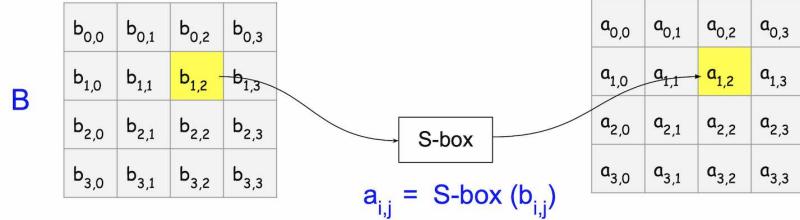


### 4. Add Round Key

Ogni byte della matrice è posto in XOR bit a bit, con un byte della chiave locale di fase

### Substitution byte

Ogni byte del blocco  $B$  è trasformato mediante una  $S$ -box:



Essa è una matrice di  $16 * 16$  byte, contenente una permutazione di tutti i 256 interi da 8 bit.

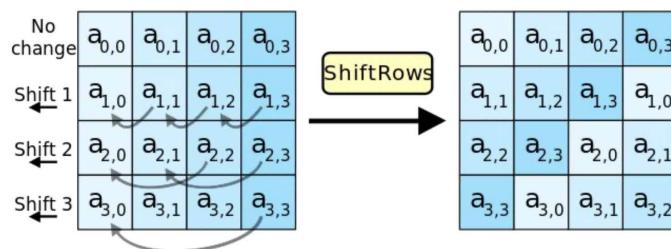
S-Box															
99	124	119	123	242	107	111	197	48	1	103	43	254	215	171	118
202	130	201	125	250	89	71	240	173	212	162	175	156	164	114	192
183	253	147	38	54	63	247	204	52	165	229	241	113	216	49	21
4	199	35	195	24	150	5	154	7	18	128	226	235	39	178	117
9	131	44	26	27	110	90	160	82	59	214	179	41	227	47	132
83	209	0	237	32	252	177	91	106	203	190	57	74	76	88	207
208	239	170	251	67	77	51	133	69	249	2	127	80	60	159	168
81	163	64	143	146	157	56	245	188	182	218	33	16	255	243	210
205	12	19	236	95	151	68	23	196	167	126	61	100	93	25	115
96	129	79	220	34	42	144	136	70	238	184	20	222	94	11	219
224	50	58	10	73	6	36	92	194	211	172	98	145	149	228	121
231	200	55	109	141	213	78	169	108	86	244	234	101	122	174	8
186	120	37	46	28	166	180	198	232	221	116	31	75	189	139	138
112	62	181	102	72	3	246	14	97	53	87	185	134	193	29	158
225	248	152	17	105	217	142	148	155	30	135	233	206	85	40	223
140	161	137	13	191	230	66	104	65	153	45	15	176	84	187	22

Tale matrice è costruita nel seguente modo: ogni byte  $b_{i,j}$  viene prima sostituito con il suo inverso moltiplicativo in  $GF(2^8)$ , e poi moltiplicato per una matrice di 8 \* 8 bit e sommato con un vettore colonna.

**Osservazione:** La  $S$ -box non ha né punti fissi né punti inversi.

### Shift Rows

I byte di ogni riga vengono shiftati ciclicamente verso sinistra di 0, 1, 2 e 3 posizioni, rispettivamente.



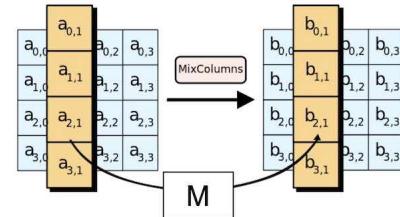
In questo modo i 4 byte di ogni colonna si disperdono su 4 colonne diverse.

### Mix Columns

Ogni colonna del blocco, trattata come un vettore di 4 elementi, viene moltiplicata per un matrice  $M$  prefissata di  $4 * 4$  byte. La moltiplicazione è eseguita mod28, e la somma mod2 (operazioni del campo  $GF(2^8)$ ).

La matrice  $M$  è scelta in modo che ciascun byte della colonna venga mappato in un nuovo byte che è funzione di tutti i 4 byte presenti nella colonna.

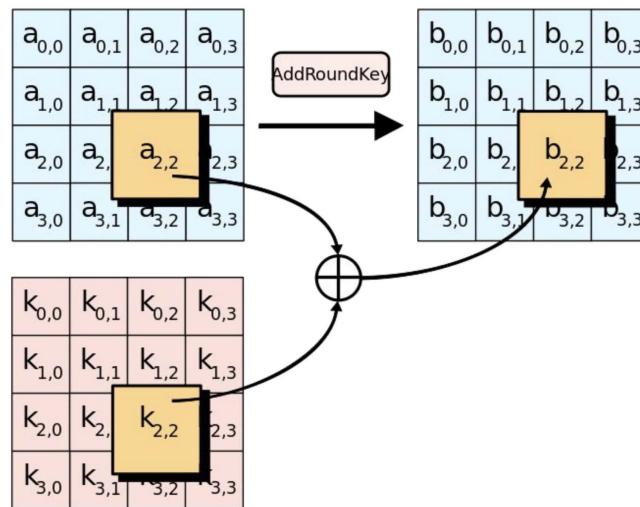
$a_{i,j}$  si trasforma in un valore  $b_{i,j}$  che dipende da tutti i byte  $a_{0,j}, a_{1,j}, a_{2,j}, a_{3,j}$  della colonna



**Osservazione:** Shift Rows e Mix Columns garantiscono diffusione totale dopo solo due fasi: ogni bit di output dipende da tutti i bit di input.

### Add Round Key

Ogni byte della matrice è posto in XOR con un byte della chiave locale di fase.



#### 11.1.4 Sicurezza

Ad oggi, nessuno attacco è stato in grado di compromettere AES anche nella sua versione più semplice, con chiave di 128 bit.

Si conoscono **side-channel attacks** che non sfruttano le caratteristiche del cifrario, ma le possibili debolezze della piattaforma su cui esso è implementato.

## 11.2 Cifrari a composizione di blocchi

Il problema dei cifrari a blocchi, come il DES (64 bit) e l'AES (128 bit), è che blocchi uguali nel messaggio producono blocchi cifrati uguali, inducendo una periodicità nel crittogramma, la quale fornisce utili informazioni per la crittoanalisi.

I metodi proposti per ovviare al problema intervengono **componendo i blocchi** tra loro. Il più diffuso tra questi metodi è il **CBC**.

### 11.2.1 CBC

Il metodo CBC (**Cipher Block Chaining**) segue il principio della diffusione di Shannon, inducendo una dipendenza di posizione tra il blocco in elaborazione e quelli precedenti. Come risultato, **blocchi uguali nel messaggio vengono** (con pratica certezza) **cifrati in modo diverso**.

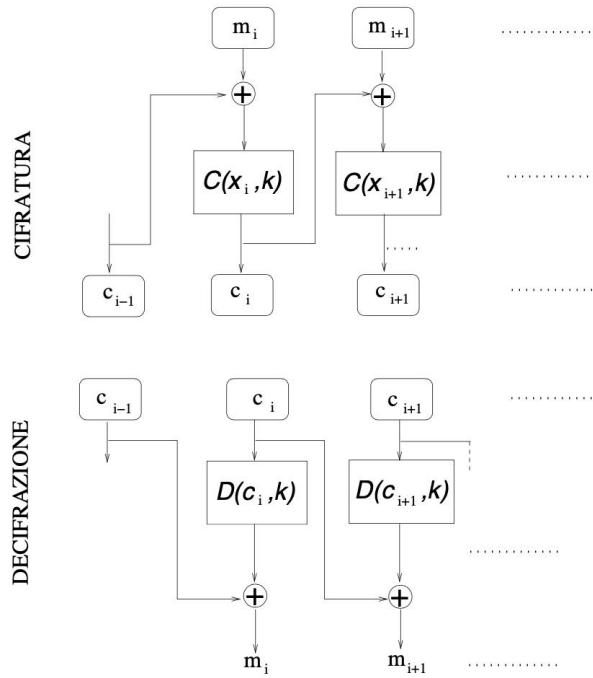
tale metodo si impiega in cifratura e in decifrazione utilizzando le funzioni  $C$  e  $D$  del cifrario d'origine.

#### Funzionamento

Supponiamo che il cifrario d'origine operi su blocchi di  $b$  bit con chiave  $k$ . Quindi, il messaggio  $m$  è diviso in blocchi  $m_1 \dots m_s$ .

Se  $m_s$  contiene  $r < b$  bit, allora lo si completa aggiungendovi la sequenza binaria 1000... di lunghezza  $b - r$ . Altrimenti, si aggiunge al messaggio un intero nuovo blocco  $m_{s+1} = 1000\dots$ . In entrambi i casi, la nuova sequenza funge da terminatore del messaggio.

Sia ora  $c_0$  una sequenza di  $b$  bit scelta in modo casuale e diversa per ogni messaggio. Il metodo opera come illustrato:



Ogni blocco  $m_i$  viene cifrato come

$$m_i \longrightarrow c_i = C(m_i \oplus c_{i-1}, k)$$

mentre la decifrazione del blocco  $c_i$  è eseguita come

$$c_i \longrightarrow m_i = c_{i-1} \oplus D(c_i, k)$$

**Osservazione:** Mentre il processo di cifratura è strettamente sequenziale, il processo di decifrazione può essere eseguito in parallelo, se tutti i blocchi cifrati sono disponibili.

# Chapter 12

## Lezione 12 (25/10/2022) Crittografia a chiave pubblica

Nei cifrari *simmetrici* visti sinora, la chiave di cifratura è uguale a quella di decifrazione, ed è nota solo ai due partner che la scelgono di comune accordo.

Nei cifrari a chiave pubblica, o **asimmetrici**, le chiavi di cifratura e di decifrazione sono **completamente diverse tra loro**. Esse sono scelte dal destinatario, che rende **pubblica** la chiave di cifratura  $k_{pub}$  e mantiene **privata** (segreta) la chiave di decifrazione  $k_{priv}$ .

In sostanza, ogni utente possiede una coppia di chiavi  $(k_{pub}, k_{priv})$ , la prima di dominio pubblico e la seconda conosciuta solo a lui. Considerando un utente *Dest* con coppia di chiavi  $((k'_{pub}, k'_{priv}))$ , la cifratura di un messaggio  $m$  da inviare a *Dest*, è eseguita da qualunque mittente come  $c = C(m, k'_{pub})$ .

**Osservazione:** Naturalmente, anche le due funzioni di cifratura  $C$  e decifrazione  $D$  sono pubbliche.

La decifrazione è eseguita da *Dest* come  $m = D(c, k'_{priv})$ , ove ricordiamo che  $k'_{priv}$  la conosce solo *Dest*, e quindi solo lui è capace di decifrare il crittogramma a lui inviato.

### 12.0.1 Funzione di cifratura

Nei cifrari asimmetrici, la funzione di cifratura  $C$ , oltre a dover ovviamente essere one-way, deve inoltre contenere un meccanismo segreto chiamato **trap-door**, che ne consenta la facile invertibilità solo a chi conosca tale meccanismo.

La conoscenza di  $k_{pub}$  non fornisce alcuna indicazione sul trap-door, il quale è svelato da  $k_{priv}$  quando questa essa è inserita nella funzione  $D$ .

## 12.1 Richiami di algebra modulare

### 12.1.1 Modulo

Preso un numero intero positivo  $n$  definiamo  $\mathcal{Z}_n = \{0, 1, \dots, n-1\}$  e  $\mathcal{Z}_n^*$  l'insieme di tutti gli elementi di  $\mathcal{Z}_n$  coprimi con  $n$ .

Il problema di determinare  $\mathcal{Z}_n^*$  dato  $n$  è generalmente computazionalmente difficile, poiché richiede tempo proporzionale al valore di  $n$ , e quindi esponenziale nella sua dimensione. Se però  $n$  è primo, allora si ha  $\mathcal{Z}_n^* = \mathcal{Z}_n \setminus \{0\}$ .

Dati tre interi  $a, b \geq 0$  e  $n > 0$ , si ha che  $a \equiv b \pmod{n}$  (letto “ $a$  congruo a  $b$  modulo  $n$ ”), se  $a \pmod{n} = b \pmod{n}$ .

**Osservazione:** Nelle relazioni di congruenza, la notazione  $\pmod{n}$  si riferisce all’intera relazione, mentre nelle relazioni di uguaglianza la stessa notazione si riferisce solo al membro ove appare.

L’operazione di **modulo** possiede le seguenti proprietà:

- $(a + b) \pmod{n} = ((a \pmod{n}) + (b \pmod{n})) \pmod{n}$
- $(a - b) \pmod{n} = ((a \pmod{n}) - (b \pmod{n})) \pmod{n}$
- $(a * b) \pmod{n} = ((a \pmod{n}) * (b \pmod{n})) \pmod{n}$
- $a^{r*s} \pmod{n} = (a^r \pmod{n})^s \pmod{n}$  con  $r, s$  interi positivi.

### 12.1.2 Funzione di Eulero

Per un intero  $n > 1$ , si definisce la funzione di Eulero  $\Phi(n)$  come il numero di interi minori di  $n$  e coprimi con esso. Quindi,  $\Phi(n) = |\mathcal{Z}_n^*|$ .

**Osservazione:**  $n$  primo  $\implies \Phi(n) = n - 1$ .

**Teorema:** Se  $p, q$  sono due numeri primi, allora si ha

$$n = p * q \implies \Phi(n) = (p - 1)(q - 1)$$

**Teorema di Eulero:**

$$\forall n > 1 \wedge a \text{ coprimo con } n . a^{\Phi(n)} \equiv 1 \pmod{n}$$

**Piccolo teorema di Fermat:** Se  $n$  è primo, allora si ha

$$\forall a \in \mathcal{Z}_n^* . a^{n-1} \equiv 1 \pmod{n}$$

Per ogni  $a$  coprimo con  $n$  si ha  $a * a^{\Phi(n)-1} \equiv 1 \pmod{n}$  per il teorema di Eulero. Inoltre,  $a * a^{-1} \equiv 1 \pmod{n}$  per la definizione di inverso. Questo implica che  $a^{\Phi(n)-1} \pmod{n} = a^{-1} \pmod{n}$ , e quindi l’inverso di  $a$  modulo  $n$  si può calcolare per esponenziazione di  $a$ , se si conosce  $\Phi(n)$ .

### 12.1.3 Equazioni in modulo

Studiamo l’equazione  $ax \equiv b \pmod{n}$ , con  $a, b, n$  interi.

**Teorema:** L'equazione  $ax \equiv b \pmod{n}$  ammette soluzione se e solo se  $MCD(a, n)$  divide  $b$ . In questo caso si hanno esattamente  $MCD(a, n)$  soluzioni distinte.

Dunque, l'equazione considerata ammette esattamente una soluzione se e solo se  $MCD(a, n) = 1$ , cioè  $a$  ed  $n$  sono coprimi.

Ponendo  $b = 1$ , la soluzione è proprio  $a^{-1}$ . Abbiamo visto che tale inverso può essere calcolato come  $a^{\Phi(n)-1} \pmod{n}$ , ma ciò richiede di conoscere  $\Phi(n)$ , richiede cioè di **fattorizzare**  $n$ . Questo è un problema computazionalmente difficile.

#### 12.1.4 Calcolo dell'inverso

Esiste un altro procedimento per il calcolo dell'inverso, basato su un'estensione dell'algoritmo di Euclide. Tale algoritmo può essere infatti generalizzato per risolvere l'equazione in due incognite  $ax + by = MCD(a, b)$ , secondo il seguente schema:

```
Function Extended_Euclid(a, b):
    if b = 0
        then return ⟨a, 1, 0⟩
    else ⟨d', x', y'⟩ ← Extended_Euclid(b, a mod b);
        ⟨d, x, y⟩ ← ⟨d', y', x' - ⌊a/b⌋y'⟩;
    return ⟨d, x, y⟩.
```

Questa funzione restituisce una delle triple di valori  $(MCD(a, b), x, y)$  con  $x, y$  tali che  $ax + by = MCD(a, b)$ .

**Osservazione:** *Extended\_Euclid* è polinomiale nella dimensione dei parametri in input.

Questa funzione permette di calcolare direttamente e efficientemente l'inverso  $x = a^{-1} \pmod{b}$  di un intero arbitrario  $a$  coprimo con  $b$ . Infatti,  $ax \equiv 1 \pmod{b}$  è equivalente a  $ax = bz + 1$  per un qualche  $z$ , da cui, ponendo  $y = -z$  e ricordando che  $MCD(a, b) = 1$ , si ottiene proprio l'equazione  $ax + by = MCD(a, b)$ , e dalla soluzione di questa si ricava il valore dell'inverso  $x$ .

#### 12.1.5 Generatori

Esistono particolari interi  $n$  ed  $a \in \mathbb{Z}_n^*$ , per cui la funzione  $a^k \pmod{n}$ , con  $k \in \{1, \dots, \Phi(n)\}$ , genera tutti e soli gli elementi di  $\mathbb{Z}_n^*$ .

**Notazione:**  $a$  è detto **generatore** di  $\mathbb{Z}_n^*$ .

**Nota:** Noi ci occuperemo solo del caso in cui  $n$  è primo, e quindi  $\mathbb{Z}_n^* = \{1, \dots, n-1\}$  e  $\Phi(n) = n-1$ .

**Teorema:** Per ogni  $n$  primo,  $\mathbb{Z}_n^*$  ammette  $\Phi(n-1)$  generatori.

Due problemi computazionali legati ai generatori sono particolarmente rilevanti in crittografia.

Il primo è quello della **determinazione di un generatore**  $a$  di  $\mathbb{Z}_n^*$ . Tale problema è ritenuto computazionalmente difficile, e viene in pratica risolto, con alta probabilità di successo, impiegando algoritmi randomizzati.

Il secondo problema, noto come **calcolo del logaritmo discreto**, riguarda la risoluzione nell'incognita  $x$  dell'equazione  $a^x \bmod n$ . Sappiamo che questa equazione ammette soluzione per ogni  $b$  se e solo se  $a$  è un generatore di  $\mathbb{Z}_n^*$ . Tuttavia, non è noto per quale valore di  $x$  si genera  $b \bmod n$ . Anche in questo caso, non è noto un algoritmo polinomiale di risoluzione del problema.

Infine, la sequenza  $a^k \bmod n$  prodotta da un generatore  $a$ , si ripete identica per  $k = \Phi(n)+1, \Phi(n)+2, \dots$

**Esempio:** Per  $n = 7$ , il generatore 3 produce la sequenza 3, 2, 6, 4, 5, 1 ripetuta all'infinito.

## 12.2 Funzioni one-way trap-door

Le funzioni one-way trap-door sono funzioni per cui:

- calcolarle è incondizionatamente semplice;
- invertirle è semplice solo se si dispone di una informazione aggiuntiva sui dati, cioè di una chiave privata. In assenza tale informazione, l'inversione richiede la soluzione di un problema NP-arduo, o comunque NP.

### 12.2.1 Esempi

Consideriamo adesso tre funzioni one-way trap-door particolarmente importanti in crittografia.

#### Fattorizzazione

Calcolare  $n = p * q$ , con  $p, q$  interi, è ovviamente facile. Invece, ricavare  $p, q$  a partire da  $n$  è possibile solamente se  $p, q$  sono **primi**. In ogni caso, tale inversione richiede tempo esponenziale. Tuttavia, la conoscenza di uno dei due fattori (chiave segreta) rende l'inversione molto facile.

**Osservazione:** Il problema della fattorizzazione non è dimostrato essere NP-arduo, anche se ad oggi non è stato trovato un algoritmo di risoluzione polinomiale.

#### Calcolo della radice in modulo

Calcolare  $y = x^z \bmod s$ , con  $x, z, s$  interi, richiede tempo polinomiale se si procede per successive esponenziazioni. Invece, se  $s$  non è primo, e se ne ignora la fattorizzazione, invertire la funzione ( $x = \sqrt[z]{y} \bmod s$ , noti  $y, z, s$ ) richiede tempo esponenziale. Tuttavia, se  $x$  è coprimo con  $s$ , e si conosce l'inverso  $v$  di  $x \bmod \Phi(s)$  ( $zv \equiv 1 \bmod \Phi(s)$ ), si ha:

$$y^v \bmod s = x^{zv} \bmod s = x^{1+k\Phi(s)} \bmod s = x \bmod s$$

ove l'ugualianza è basata sul teorema di Eulero.

Quindi, si ricava  $x$  calcolando  $y^v \bmod s$  in tempo polinomiale. In questo caso,  $v$  è la chiave segreta.

**Osservazione:** Il problema del calcolo della radice in modulo non è dimostrato essere NP-arduo, anche se ad oggi non è stato trovato un algoritmo di risoluzione polinomiale.

### Calcolo del logaritmo discreto

Dati  $x, y, s$  interi, si richiede di trovare il valore  $z$  per il quale  $y = x^z \pmod{s}$ . Operando in modulo, il problema è computazionalmente difficile e non sempre ha soluzione. Se  $s$  è primo, allora esiste una soluzione per ogni  $y$  se e solo se  $x$  è un generatore di  $\mathbb{Z}_s^*$ . A parte alcuni particolari primi  $s$  per cui il calcolo si semplifica, ad oggi gli algoritmi di risoluzione hanno la stessa complessità di quelli per la fattorizzazione, quindi esponenziale.

Ciò dimostra che la funzione di calcolo del logaritmo discreto è one-way. È inoltre possibile dimostrare, con un artificio piuttosto complesso, come introdurre una trap-door in essa.

## 12.3 Pregi e difetti

Due vantaggi immediati della crittografia a chiave pubblica, rispetto a quella simmetrica, sono:

- Dato un sistema con  $n$  utenti, il numero complessivo di chiavi (pubbliche e private) è  $2n$ , anziché  $n(n - 1)/2$ .
- Non è richiesto alcuno scambio segreto di chiavi tra gli utenti.

Tuttavia, il mondo non è tutto rose e fiori, ed anche questo sistema presenta significativi svantaggi:

- Il sistema è ovviamente esposto a chosen plain-text attacks, dato che un crittoanalista può cifrare con la chiave pubblica di utente tutti i messaggi che vuole. In questo modo, spiando il canale in entrata dell'utente, può sapere se riceve oppure *non* riceve ceterminati messaggi. Questo attacco è particolarmente pericoloso se il crittoanalista sospetta che l'utente debba ricevere un messaggio particolare ed è in attesa di vedere quando questo accada.
- Questo sistema è molto più lento di quelli basati su cifrari simmetrici, di due o tre ordini di grandezza.

### Approccio ibrido

Poichè i cifrari simmetrici e asimmetrici presentano pregi e difetti complementari, nei protocolli crittografici esistenti si tende ad adottare un approccio ibrido, che combina i pregi ed esclude i difetti di ognuno.

Infatti, spesso si usa un cifrario a chiave segreta come il Triplo DES o l'AES per le comunicazioni di massa, e un cifrario a chiave pubblica per scambiare le chiavi segrete relative al primo, senza un incontro fisico tra gli utenti.

In questo modo si risolve anche il problema dei chosen plain-text attacks, dato che l'unica informazione scambiata col cifrario asimmetrico è la chiave del cifrario simmetrico, chiave generata casualmente.

# **Chapter 13**

## **Lezione 13 (31/10/2022)**

Sono stati fatti solo esercizi. Quando li rifarò li metterò qui.

# Chapter 14

## Lezione 14 (07/11/2022) RSA

RSA, nato ufficialmente nel 1978, è stato tra i cifrari a chiave pubblica più impiegati negli anni. Questo è stato dovuto alla sua estrema semplicità strutturale. Inoltre, il cifrario è tutt'ora risultato inviolato.

### 14.1 Funzionamento

Vediamo a questo punto struttura e funzionamento generale dell'RSA.

#### 14.1.1 Creazione delle chiavi

Il destinatario *Dest* esegue le seguenti operazioni:

- sceglie due numeri primi  $p, q$  molto grandi. Ciò può essere fatto in tempo polinomiale con l'algoritmo di Miller e Rabin.
- calcola  $n = p * q$ . Si noti che  $p, q$  devono essere abbastanza grandi affinché  $n$  abbia almeno **2048** cifre binarie, per essere considerato sicuro fino al 2030, altrimenti almeno **3072**.
- calcola  $\Phi(n) = (p - 1) * (q - 1)$ .
- sceglie un intero  $e < \Phi(n)$  e coprimo con esso.
- calcola  $d = e^{-1} \bmod \Phi(n)$ . Il calcolo si esegue polinomialmente tramite Euclide esteso.

A questo punto, otterremo le chiavi:

- $k_{pub} = (e, n)$
- $k_{priv} = d$

#### 14.1.2 Messaggio

Ogni messaggio è codificato in una sequenza binaria, che viene trattata come un **numero intero**  $m$ . Per poter utilizzare il cifrario, bisogna avere  $m < n$ . Quindi, un messaggio troppo lungo lo si può dividere in blocchi di al più  $\log_2 n$  bit, cifrati indipendentemente.

**Osservazione:** Si noti che se  $m > n$ , allora il cifrario non sarebbe più iniettivo, dato che  $m$  e  $m \bmod n$  genererebbero lo stesso crittogramma.

### 14.1.3 Cifratura e decifrazione

Il crittogramma  $c$  viene calcolato da  $m$  nel seguente modo:

$$c = m^e \bmod n$$

A questo punto, si ricava invece  $m$  da  $c$  nel seguente modo:

$$m = c^d \bmod n$$

**Osservazione:** Cifratura e decifrazione si calcolano per successive esponenziazioni in tempo polinomiale.

#### Esempio

Scegliamo  $p = 5, q = 11$ . Quindi  $n = 55$  e  $\Phi(n) = 40$ . A questo punto, scegliamo  $e = 7$ . Calcoliamo quindi  $d = 23$  con Euclide esteso.

Abbiamo ottenuto:

- $k_{pub} = (7, 55)$
- $k_{priv} = 23$

Per cifrare un messaggio  $m < 55$ , calcoleremo  $c = m^7 \bmod 55$ . La decifrazione sarà perciò  $m = c^{23} \bmod 55$ .

## 14.2 Correttezza

#### Teorema:

$$\forall m < n . (m^e \bmod n)^d \bmod n = m$$

ove  $n, e, d$  sono i parametri dell'RSA.

#### Dimostrazione

Si noti anzi tutto che

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$

per proprietà dell'algebra modulare. Detto ciò, per la dimostrazione distinguiamo i due casi seguenti.

[  **$p$  e  $q$  non dividono  $m$**  ] In questo caso,  $m$  ed  $n$  sono **coprimi**. Per il teorema di Eulero, risulta quindi  $m^{\Phi(n)} \equiv 1 \bmod n$ . Poiché  $d = e^{-1} \bmod \Phi(n)$ , abbiamo che  $e * d \equiv 1 \bmod \Phi(n)$ , cioè  $e * d = 1 + r\Phi(n)$ , con  $r$  un certo intero positivo. Si ottiene quindi:

$$m^{ed} \bmod n = m^{1+r\Phi(n)} \bmod n = m * (m^{\Phi(n)})^r \bmod n = m * 1^r \bmod n = m$$

[  **$p$  o  $q$  divide  $m$**  ] Supponiamo  $p \mid m$  ( $p$  divide  $m$ ) e  $q \nmid m$  ( $q$  non divide  $m$ ). A questo punto, abbiamo  $m \equiv m^r \equiv 0 \pmod{p}$ , cioè  $(m^r - m) \equiv 0 \pmod{p}$ , per ogni intero positivo  $r$ . Abbiamo inoltre:

$$\begin{aligned} m^{ed} \pmod{q} &= m^{1+r\Phi(n)} \pmod{q} = m * (m^{r(p-1)(q-1)}) \pmod{q} = \\ &= m * (m^{q-1})^{r(p-1)} \pmod{q} = m * 1^{r(p-1)} \pmod{q} = m \pmod{q} \end{aligned}$$

Dunque,  $m^{ed} \equiv m \pmod{q}$ , cioè  $(m^{ed} - m) \equiv 0 \pmod{q}$ . Ne consegue che  $(m^{ed} - m) \equiv 0 \pmod{n}$ , ove ricordiamo che  $n = p * q$ . Da qui è immediato che:

$$m^{ed} \pmod{n} = m \pmod{n} = m$$

### 14.3 Sicurezza

La sicurezza dell'RSA è strettamente legata alla difficoltà di fattorizzare un numero intero arbitrario molto grande. Tale problema può essere risolto in tempo subesponenziale con l'algoritmo GNFS. Infatti, si noti che la chiave privata  $d = e^{-1} \pmod{\Phi(n)}$  può essere indirettamente scoperta soltanto fattorizzando  $n$ , per coprire i  $p$  e  $q$ .

Si può pensare di calcolare  $\Phi(n)$  direttamente senza fattorizzare, ma le due cose sono equivalenti! (more on this later)

Si potrebbe anche pensare di bypassare il calcolo della chiave privata, estrapolando direttamente il messaggio  $m$  dal crittogramma  $c$  come  $m = \sqrt[p]{c} \pmod{n}$ . Tuttavia, il calcolo della radice in modulo, con  $n$  composto, è difficile quanto la fattorizzazione.

#### Fattorizzazione vs Calcolo diretto

L'equivalenza dei due problemi si dimostra facilmente in due passaggi:

- Fattorizzato  $n$ , abbiamo trovato  $p$  e  $q$ . Da qui è immediato trovare  $\Phi(n) = (p-1)*(q-1)$ .
- Calcolato direttamente  $\Phi(n)$  da  $n$ , è immediato trovare  $p$  e  $q$ , nel seguente modo:
  - $\Phi(n) = (p-1)*(q-1) = pq - (p+q) + 1 = n - (p+q) + 1$
  - $(p+q) = n - \Phi(n) + 1$
  - $(p-q)^2 = (p+q)^2 - 4pq = (p+q)^2 - 4n$
  - $p = \frac{(p+q)+(p-q)}{2}$
  - $q = \frac{(p+q)-(p-q)}{2}$

**Osservazione:** Fattorizzazione e calcolo del logaritmo discreto **non** sono problemi NP-hard, e si possono risolvere in tempo polinomiale su macchine quantistiche.

Insomma, si capisce che la sicurezza dell'RSA aumenta all'aumentare dei bit di  $n$ . Di seguito, si può osservare una tabella che compara il numero di bit di  $n$  nell'RSA e di bit della chiave segreta nell'AES che garantiscono lo stesso livello di sicurezza.

TDEA, AES (bit della chiave)	RSA e DH (bit del modulo)
80	1024
112	2048
128	3072
192	7680
256	15360

### 14.3.1 Vincoli

#### Vincoli su p e q

Vediamo quali sono i vincoli nella scelta di  $p$  e  $q$ :

- $p, q$  **grandi** per evitare attacchi bruteforce.
- Deve essere grande la loro **differenza**. Infatti, se  $|p - q|$  fosse piccolo, allora  $(p + q)/2 \approx \sqrt{n}$ . Ciò renderebbe facile trovare  $p$  e  $q$ , nel seguente modo:
  - Anzitutto vale l'ugualanza:

$$\frac{(p+q)^2}{4} - n = \frac{(p-q)^2}{4}$$

che mostra come  $\frac{(p-q)^2}{4}$  sia un quadrato perfetto.

- È immediato dimostrare che:

$$\frac{p+q}{2} > \sqrt{n}$$

- Si possono perciò scandire gli interi maggiori di  $\sqrt{n}$  fino a trovare  $z$  tale che  $w^2 = z^2 - n$  sia un quadrato perfetto.
- Si suppone quindi  $z = \frac{p+q}{2}$  e  $w = \frac{p-q}{2}$ , da cui si inferiscono i valori  $p = z + w$  e  $q = z - w$ .
- Si controlla quindi la significatività di tali valori su  $c$ .

A questo punto, la vicinanza di  $p$  e  $q$  implica che non bisogna fare molti tentativi prima di scoprirli.

- Sia  $(p-1)$  che  $(q-1)$  devono contenere un **fattore primo grande**.
- $\frac{p-1}{2}$  e  $\frac{q-1}{2}$  devono essere **coprimi**.

#### Vincoli su e

- Vi sono vincoli anche sulla scelta di  $e$ , in particolare: se  $k$  divide sia  $(p-1)$  che  $(q-1)$ , e se  $m$  ed  $n$  sono coprimi, allora si può dimostrare che:

$$e = 1 + \Phi(n)/k \implies c = m^e \bmod n = m$$

Quindi il crittogramma sarebbe identico al messaggio.

- Inoltre, se almeno  $e$  utenti hanno lo stesso  $e$  e ricevono lo stesso messaggio  $m$ , attraverso i crittogrammi

$$c_i = m^e \bmod n_i \quad \forall i \in \{1, \dots, e\}$$

allora, per il **teorema cinese del resto**:

$$\exists! m' < n = \prod_{i=1}^e n_i : m' \equiv m^e \bmod n = m^e \implies m = \sqrt[e]{m'}$$

e siccome  $m'$  si può calcolare in tempo polinomiale, allora anche  $m$  si può trovare facilmente.

- Infine, è fondamentale che gli utenti scelgano valori di  $n$  diversi fra loro, anche se le chiavi potrebbero essere differenziate con la scelta di  $e$ . In caso contrario, un crittoanalista potrebbe selezionare due coppie di chiavi  $(e_1, n), (e_2, n)$  tali che  $MCD(e_1, e_2) = 1$ . Quindi, per l'**identità di Bézout**, esistono, calcolabili con Euclide esteso, due interi  $r, s$  per cui  $e_1r + e_2s = 1$ .

Supponiamo ora che  $r < 0$  e che il crittoanalista abbia due crittogrammi  $c_1, c_2$  relativi allo stesso messaggio  $m$  inviato ai due utenti. Abbiamo allora:

$$m = m^{e_1r+e_2s} = (c_1^r * c_2^s) \bmod n = ((c_1^{-1})^{-r} * c_2^s) \bmod n$$

Poiché, in pratica,  $c_1$  ed  $n$  sono coprimi, il crittoanalista può calcolare  $c_1^{-1} \bmod n$  con Euclide esteso, e quindi  $(c_1^{-1})^{-r}$  per esponenziazione. Infine, può calcolare anche  $c_2^s$  per esponenziazione, cosicché può facilmente ricostruire  $m$ .

### 14.3.2 Attacchi a tempo

Esistono anche attacchi che si basano sul tempo d'esecuzione dell'algoritmo di decifrazione. L'idea è quella di determinare  $d$  analizzando il tempo impiegato per decifrare.

Questo è possibile in quanto quando viene eseguita l'esponenziazione modulare, si esegue una moltiplicazione ad ogni iterazione, più un'ulteriore moltiplicazione modulare per ciascun bit uguale a 1 in  $d$ .

In questo caso, il rimedio è semplice e consiste nell'aggiungere un ritardo casuale alla terminazione del programma.

# Chapter 15

## Lezione 15 (09/11/2022) ElGamal, DH

Iniziamo guardando un altro cifrario asimmetrico: il cifrario di **ElGamal**. Esso basa la sua sicurezza sulla difficoltà del **calcolo di logaritmi discreti**, mentre RSA si affida invece alla difficoltà di fattorizzare numeri grandi.

### 15.1 Cifrario di ElGamal

#### 15.1.1 Funzionamento

Vediamo a questo punto struttura e funzionamento generale di ElGamal.

##### Creazione delle chiavi

Il destinatario *Dest* esegue le seguenti operazioni:

- sceglie un numero primo  $p$  **molto grande**
- sceglie un generatore  $g$  per  $\mathbb{Z}_p^*$
- sceglie un  $x \in \{2, \dots, p - 2\}$
- calcola  $y = g^x \bmod p$

A questo punto, otterremo le chiavi:

- $k_{pub} = (p, g, y)$
- $k_{priv} = x$

##### Messaggio

Ogni messaggio è codificato in una sequenza binaria, che viene trattata come un numero intero  $m$ . Per poter utilizzare il cifrario, bisogna avere  $m < p$ . Quindi, un messaggio troppo lungo lo si può dividere in blocchi di al più  $\log_2 p$  bit, cifrati indipendentemente.

### Cifratura e decifrazione

La cifratura del messaggio  $m$  genera una **coppia di crittogrammi**  $(c, d)$ , calcolati nel seguente modo:

- Si sceglie a caso  $r \in \{2, \dots, p - 2\}$
- $c = g^r \bmod p \quad - \quad d = m * y^r \bmod p$

A questo punto,  $m$  si ricava nel seguente modo:

$$m = d * c^{-x} \bmod p$$

#### 15.1.2 Correttezza

La correttezza del cifrario è di dimostrazione immediata, infatti:

$$d * c^{-x} \bmod p = m * y^r * c^{-x} \bmod p = m * (g^x)^r * (g^r)^{-x} \bmod p = m * g^{xr} * g^{-xr} \bmod p = m$$

#### 15.1.3 Sicurezza

Come già introdotto, il cifrario di ElGamal si affida alla difficoltà del calcolo del logaritmo discreto. Infatti, supponiamo che il crittoanalista conosca  $p, g, y, c, d$ , cioè chiave pubblica e coppia di crittogrammi:

- se conosce  $x$  (che non deve conoscere), allora calcola immediatamente  $m = d * c^{-x} \bmod p$ .
- se non conosce  $x$  (come deve essere) può trovarlo calcolando  $x = \log_g y$ , che è un logaritmo **discreto**. Questo calcolo richiede tempo sub-esponenziale, e quindi ci va bene.
- se conosce  $r$  (che non deve conoscere), allora trova  $m = d * y^{-r} \bmod p$  in tempo polinomiale.

**Osservazione:** Si capisce l'importanza di mantenere **assolutamente segreti** sia  $x$  che  $r$ , dato che altrimenti un crittoanalista può ricavare il messaggio facilmente.

## 15.2 Protocollo DH

**DH (Diffie-Hellman)** è un protocollo di **scambio delle chiavi** relative ad un cifrario simmetrico. Quindi, esso **non** è un cifrario, ma solo un protocollo che permette lo scambio sicuro di chiavi **sopra un canale insicuro**.

### 15.2.1 Cifrari ibridi

Al posto di usare uno specifico protocollo di scambio delle chiavi, due utenti possono sfruttare un **cifrario ibrido**, il quale combina due cifrari: uno asimmetrico per lo scambio iniziale delle chiavi, ed uno simmetrico per lo scambio di messaggi.

Uno dei più faosi cifrari ibridi è dato dalla coppia (RSA, AES). Ogni scambio di messaggi in questo tipo di cifrario, è dato dalla coppia

$$(C_{RSA}(k_{session}, k_{pub}), C_{AES}(m, k_{session}))$$

**Osservazione:** La stessa chiave  $k_{session}$  potrebbe essere utilizzata in più comunicazioni successive, al fine di ammortizzare il costo di utilizzazione del cifrario asimmetrico. Tuttavia, la sicurezza del protocollo (cifrario ibrido) aumenta notevolmente se  $k_{session}$  cambia ad ogni scambio di messaggi.

### Svantaggi

Il problema serio dei cifrari ibridi è che la **responsabilità di creazione della chiave di sessione ricade interamente sul mittente**.

Infatti, in ambienti distribuiti come Internet, è bene attribuire pari responsabilità a mittente e destinatario per la creazione della chiave, dato che nella maggioranza dei casi non si conoscono e potrebbero non fidarsi l'uno dell'altro. Il fatto che ognuno partecipi attivamente alla creazione della chiave, permette ad entrambi di avere garanzia che la chiave sia sicura.

### 15.2.2 Protocollo DH

Il protocollo DH permette proprio uno scambio equo delle chiavi, in cui entrambi gli utenti partecipano attivamente alla loro creazione.

#### Funzionamento

Tale protocollo sfrutta l'intrattabilità del problema del calcolo del logaritmo discreto. Infatti, la creazione della chiave avviene nel seguente modo:

- Gli utenti A e B si accordano **pubblicamente** su un numero primo  $p$  molto grande, e su un generatore  $g$  di  $\mathbb{Z}_p^*$ . La coppia  $(p, g)$  può essere generata da zero, oppure presa da un insieme di coppie pubbliche.
- A sceglie un intero positivo casuale **segreto**  $x < p$ . Quindi, calcola  $X = g^x \bmod p$ , e lo spedisce **pubblicamente** a B.
- B sceglie un intero positivo casuale **segreto**  $y < p$ . Quindi, calcola  $Y = g^y \bmod p$ , e lo spedisce **pubblicamente** ad A.
- A riceve Y e calcola  $k_{session} = Y^x \bmod p (= g^{xy} \bmod p)$ .
- B riceve X e calcola  $k_{session} = X^y \bmod p (= g^{xy} \bmod p)$ .

Alla fine del protocollo, entrambi gli utenti hanno quindi generato la stessa chiave di sessione.

**Osservazione:** Si noti come B **non** conosca  $x$  ed A **non** conosca  $y$ .

#### Sicurezza

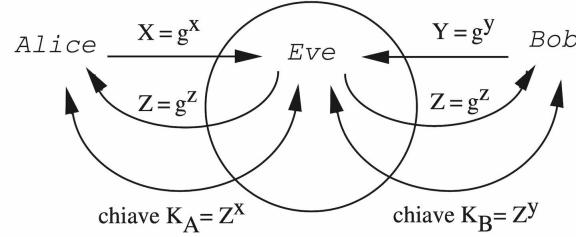
Un crittoanalista **passivo** può aver intercettato  $p, g, X$  e  $Y$ . Tuttavia, per calcolare la chiave di sessione, deve risolvere una delle due equazioni:

- $X = g^x \bmod p$  rispetto ad  $x$
- $Y = g^y \bmod p$  rispetto ad  $y$

Questo implica che il crittoanalista deve calcolare il logaritmo discreto di  $X$  od  $Y$ , problema di complessità sub-esponenziale.

## Attacchi

Nonostante il protocollo sia sicuro contro crittoanalisti passivi, un crittoanalista **attivo** (E) che sferra un attacco **man-in-the-middle** rappresenta un serio problema alla sicurezza del sistema. Infatti, se esso riesce a modificare la comunicazione tra A e B, allora può fingersi B agli occhi di A, e viceversa. Questo comporta che A parlerà con E pensando che sia B, e viceversa.



# Chapter 16

## Lezione 16 (14/11/2022) Curve ellittiche

Le curve ellittiche sono particolarmente importanti in crittografia, in quanto offrono un sistema efficiente di crittografia asimmetrica.

Come sappiamo, gli algoritmi di cifratura basati su operazioni dell'algebra modulare (RSA, ElGamal, DH) richiedono tempi di cifratura e decifrazione notevoli, specialmente se confrontati con i cifrari simmetrici. Inoltre, il progressivo aumento dei bit delle chiavi per garantire l'adeguata sicurezza del cifrario, rende impegnativa già la sola generazione della chiave stessa!

**Osservazione:** Ricordiamo che per garantire la sicurezza dell'AES a 256 bit, l'RSA richiede una chiave di oltre 15000 bit.

L'idea a questo punto è di sostituire le operazioni basate sull'algebra modulare con operazioni definite su **punti di una curva ellittica**, al fine di rendere gli algoritmi crittografici computazionalmente più leggeri.

In questa lezione ci limitiamo a vedere **cosa** sono le curve ellittiche.

### 16.1 Struttura generale

Una curva ellittica su un  è definita come l'insieme:

$$E = \{(x, y) \in K^2 : y^2 + axy + by = x^3 + cx^2 + dx + e\}$$

ove  $a, b, c, d, e \in K$ .

Tuttavia, se la **caratteristica** di  $K$  è diversa da 2 e da 3, allora l'equazione che definisce una curva ellittica si può ridurre ad una **equazione cubica in forma normale di Weierstrass**:

$$E = \{(x, y) \in K^2 : y^2 = x^3 + ax + b\}$$

**Osservazione:** La caratteristica di un campo è il più piccolo numero di volte che l'1 del campo (*identità moltiplicativa*) deve essere sommato a sé stesso per restituire lo 0 del campo (**identità additiva**). Se non esiste tale numero, allora si dice che il campo ha caratteristica 0.

**Esempio:** L'insieme dei numeri reali ha caratteristica 0. Infatti:

$$\nexists n \in \mathbb{N} : \sum_1^n 1 = 0;$$

Al contrario, l'insieme dei numeri modulo  $n$  ha caratteristica  $n$ . Infatti:

$$\sum_1^n 1 \bmod n = n \bmod n = 0$$

□

È possibile attribuire all'insieme dei punti di una curva ellittica la struttura algebrica di **gruppo abeliano additivo**, ovvero è possibile definire una legge di composizione interna che permette di associare ad ogni coppia di punti sulla curva, un terzo punto sulla curva.

## 16.2 Curve ellittiche sui reali

Le curve ellittiche sono curve algebriche descritte da **equazioni cubiche**:

$$E(a, b) = \{(x, y) \in \mathbb{R}^2 : y^2 = x^3 + ax + b\}$$

$E(a, b)$  contiene il **punto all'infinito**  $O$  in direzione dell'asse  $y$ . Ciò significa che la curva ha un asintoto verticale.

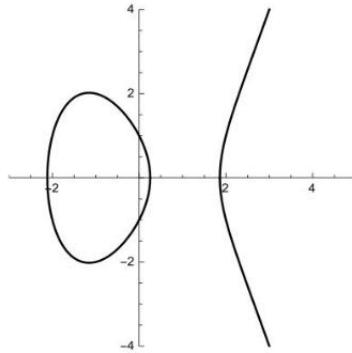
**Osservazione:** Il punto all'infinito è un **elemento neutro** per l'operazione di addizione.

### 16.2.1 Andamento

La cubica che descrive la curva può avere 3 radici reali, oppure 1 reale e 2 complesse coniugate. A seconda dei casi, la curva può assumere due principali andamenti.

#### 3 radici reali

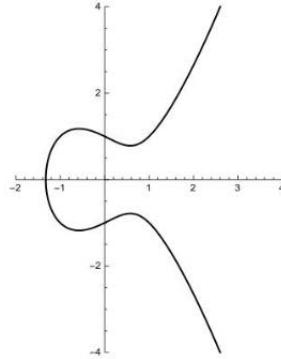
La curva ha ovviamente 3 zeri, ottenendo quindi un andamento del genere:



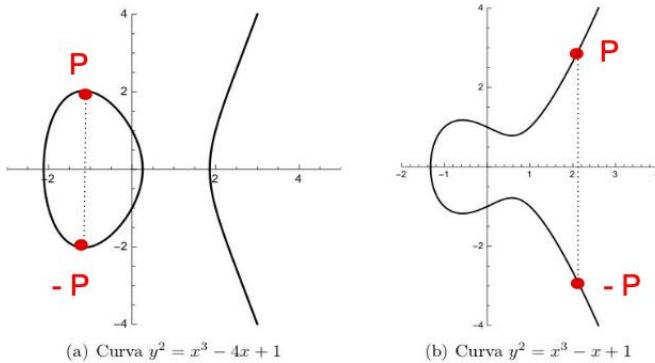
Come si può osservare, la curva è “divisa in due sottocurve disgiunte”, una chiusa ed una aperta.

## 2 radici complesse coniugate

La curva ha ovviamente un solo zero reale, ottenendo quindi un andamento del genere:



In ogni caso, è immediato osservare che le curve ellittiche presentano **simmetria orizzontale**.



E ciò non stupisce, visto che abbiamo un  $y^2$  nell'equazione che le descrive.

Quindi, ogni punto  $P = (x, y)$  sulla curva si riflette rispetto all'asse delle ascisse nel punto  $-P = (x, -y)$ , anch'esso sulla curva. Inoltre,  $O = -O$ , dato che è l'elemento neutro per l'addizione.

### 16.2.2 Vincoli crittografici

Per le applicazioni in ambito crittografico, si considerano curve ellittiche per cui  $4a^3 + 27b^2 \neq 0$ .

Questo vincolo assicura che:

- il polinomio  $x^3 + ax + b$  non abbia **radici multiple**.
- la curva sia priva di punti singolari come cuspidi o nodi; punti in cui non sarebbe definita in modo univoco la tangente.

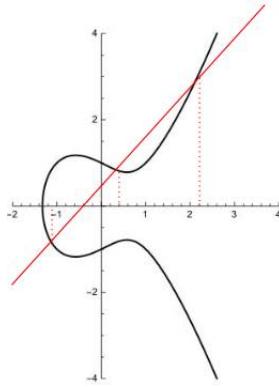
### 16.2.3 Intersezione con retta

Proprietà fondamentale delle curve ellittiche è che **una retta interseca una curva ellittica in al più tre punti**.

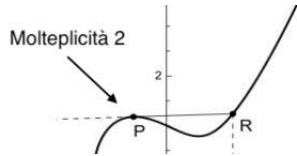
In particolare, l'intersezione tra una curva ellittica ed una retta può dare 3 soluzioni reali od una reale e due complesse coniugate.

### 3 soluzioni reali

La retta interseca la curva in 3 punti.

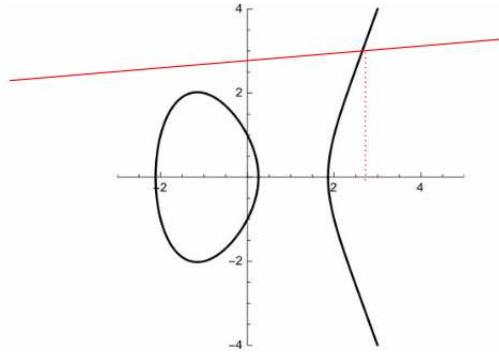


**Osservazione:** Si noti che se la retta interseca in due punti la curva, il punto a cui è tangente ha molteplicità 2, quindi la curva è comunque intersecata in 3 punti.



### 2 soluzioni complesse coniugate

La retta interseca la curva in 1 punto.



Quindi, se una retta interseca la curva  $E(a, b)$  in due punti  $P$  e  $Q$ , coincidenti se la retta è una tangente, allora la retta interseca la curva anche in un terzo punto  $R$ .

### 16.2.4 Addizione di punti

Dati tre punti  $P, Q, R \in E(a, b)$  disposti su una retta, si pone

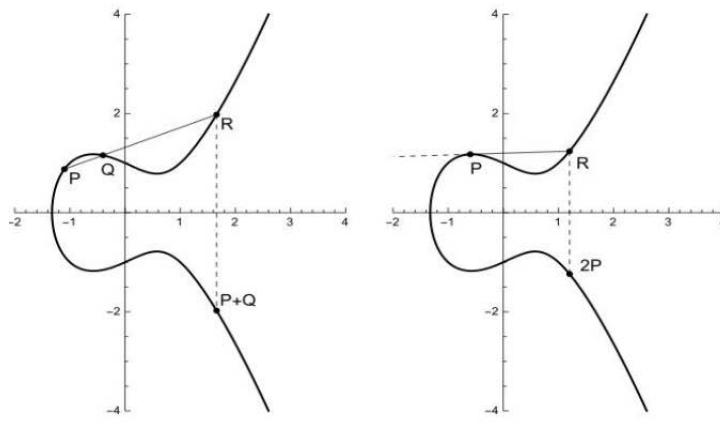
$$P + Q + R = O$$

e da ciò si ricava il concetto di somma di punti.

Infatti, per determinare la somma tra due punti  $P$  e  $Q$

1. si considera la retta passante per  $P$  e  $Q$ , oppure la retta tangente la curva in  $P$  se i due punti coincidono.
2. si determina il terzo punto di intersezione  $R$  tra la curva e la retta.
3. si definisce somma di  $P$  e  $Q$  il punto **simmetrico** a  $R$  rispetto all'asse delle ascisse.

$$P + Q = -R$$



**Osservazione:** La somma è ben definita in quanto  $-R$  è un punto della curva.

Inoltre, considerando  $P = (x_P, y_P), Q = (x_Q, y_Q)$ , si distinguono i seguenti casi:

- $P \neq \pm Q \implies S = P + Q$

$$\begin{aligned} x_S &= \lambda^2 - x_P - x_Q \\ y_S &= -y_P + \lambda(x_P - x_S) \\ \lambda &= \frac{(y_Q - y_P)}{(x_Q - x_P)} \sim \text{coefficiente angolare della retta per } P \text{ e } Q \end{aligned}$$

- $P = Q \implies S = P + Q = 2P$

$$\begin{aligned} x_S &= \lambda^2 - x_P - x_Q \\ y_S &= -y_P + \lambda(x_P - x_S) \\ \lambda &= \frac{(3x_P^2 + a)}{2y_P} \sim \text{coefficiente angolare della tangente alla curva in } P \ (y_P \neq 0) \end{aligned}$$

Se  $y_P = 0$ ,  $\textcolor{red}{S = 2P = O}$

- $P = -Q \implies S = P + Q = P + (-P) = O$

**Osservazione:** La condizione  $4a^3 + 27b^2 \neq 0$  fa sì che la legge di addizione attribuisca all'insieme dei punti di una curva ellittica  $E(a, b)$  la struttura algebrica di un **gruppo abeliano additivo**, che ha per elemento neutro il punto all'infinito  $O$ .

### Proprietà

L'addizione gode delle seguenti proprietà:

- **chiusura:**  $\forall P, Q \in E(a, b) . P + Q \in E(a, b)$
- **elemento neutro:**  $\forall P \in E(a, b) . P + O = O + P = P$
- **inverso:**  $\forall P \in E(a, b) . \exists! Q = -P \in E(a, b) : P + Q = Q + P = O$
- **associatività:**  $\forall P, Q, R \in E(a, b) . P + (Q + R) = (P + Q) + R$
- **Commutatività:**  $\forall P, Q \in E(a, b) . P + Q = Q + P$

## 16.3 Curve ellittiche su campi finiti

Le curve ellittiche su campi finiti vengono usate in ambito crittografico. Vediamo due principali famiglie di curve su campi finiti: le curve **prime** e le curve **binarie**.

I protocolli crittografici su curve ellittiche si applicano indistintamente a entrambe queste famiglie di curve.

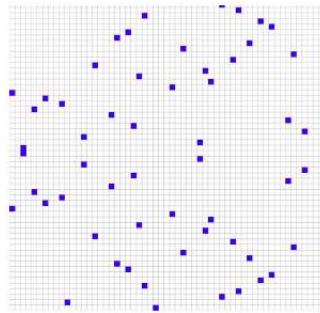
### 16.3.1 Curve ellittiche prime

Una curva ellittica prima ha variabili e coefficienti ristretti agli elementi del campo  $Z_p$ , ove  $p > 3$  è un **numero primo**. Tali curve sono formate dall'insieme:

$$E(a, b) = \{(x, y) \in Z_p^2 : y^2 \bmod p = (x^3 + ax + b) \bmod p\} \cup \{O\}$$

un tale tipo di curva possiede un numero finito di punti.

**Esempio:**

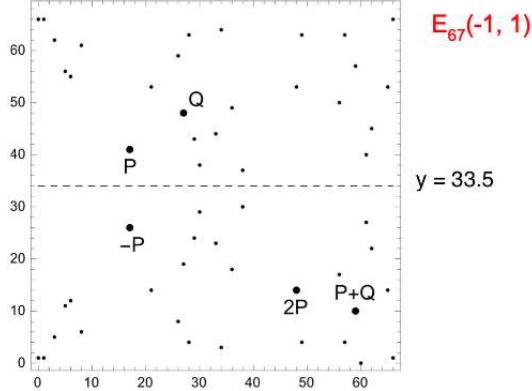


### Simmetria

Siccome una curva ellittica prima non è rappresentata nel piano, è simmetrica rispetto alla retta  $y = p/2$ . Questo implica che dato  $P = (x, y)$ , il suo inverso sarà  $-P = (x, p - y)$ .

Inoltre, se vale  $(4a^3 + 27b^2) \bmod p \neq 0$ , allora i punti di  $E(a, b)$  formano un **gruppo abeliano additivo finito**.

Esempio:



### 16.3.2 Curve ellittiche binarie

Una curva ellittica binaria ha variabili e coefficienti ristretti agli elementi del campo  $GF(2^m)$ , costituito da  $2^m$  elementi, che si possono pensare come tutti gli interi binari di  $m$  cifre, sui quali si può operare mediante l'aritmetica polinomiale modulare.

**Osservazione:** La caratteristica di  $GF(2^m)$  è ovviamente 2, quindi non si può usare la forma normale di Weierstrass per definire una curva ellittica su questo campo.

## **Chapter 17**

# **Lezione 17 (15/11/2022)**

## **Cyberchallenge**

È stata presentata la cyberchallenge. Fine.

# Chapter 18

## Lezione 18 (22/11/2022) Crittografia su curve ellittiche

Prima di parlare di crittografia su curve ellittiche, finiamo il discorso su quest'ultime.

### 18.1 Ordine di una curva ellittica

L'ordine di una curva è data dal **numero dei punti** che la compongono.

Una curva prima  $E_p(a, b)$  può avere al più  $2p + 1$  punti:

- il punto all'infinito  $O$
- $p$  coppie di punti  $(x, y), (x, -y)$  tali che  $y^2 \bmod p = (x^3 + ax + b) \bmod p$

Una curva prima contiene mediamente  $\theta(p)$  punti, considerando i residui quadratici in  $Z_p$ , cioè gli elementi del campo che ammettono radice quadrata in  $Z_p$ .

In genere non tutti i valori di  $x$  in  $Z_p$  danno luogo a un residuo quadratico nel campo, e quindi la curva non contiene punti con quelle ascisse.

**Esempio:** Se consideriamo la curva  $E_5(2, 1)$ , allora essa sarà costituita dai punti:

- $(0, 1) \quad (0, 4)$
- $(1, 2) \quad (1, 3)$
- $(2, 0)$
- $(3, 2) \quad (3, 3)$

Si noti come la curva non abbia punti con ascissa 4, dato che  $(4^3 + 2 * 4 + 1) \bmod 5 = 73 \bmod 5 = 3$ , il quale non è un residuo quadratico in  $Z_5$ . Inoltre, si noti come i punti che invece la compongono siano simmetrici rispetto alla retta  $y = 5/2$ .

Si può dimostrare che esattamente  $\frac{p-1}{2}$  elementi di  $Z_p$  (oltre lo 0) siano residui quadratici, e ciascuno di essi ha esattamente due radici quadrate in  $Z_p$ , la cui somma è proprio  $p$ . Inoltre, valori diversi di  $x$  possono generare lo stesso residuo quadratico, e quindi possono esistere molteplici punti con stessa ordinata.

**Osservazione:** Nell'esempio precedente, abbiamo i seguenti residui quadratici:

- $0 \rightarrow 0$
- $1 \rightarrow 1$
- $2 \rightarrow 4$
- $3 \rightarrow 4$
- $4 \rightarrow 1$

In generale, ci si può aspettare che la curva sia formata da circa  $p$  punti, oltre al punto all'infinito.

**Teorema (di Hasse):** L'ordine  $N$  di una curva ellittica  $E_p(a, b)$  verifica la diseguaglianza

$$|N - (p + 1)| \leq 2\sqrt{p}$$

## 18.2 Funzioni one-way trap-door

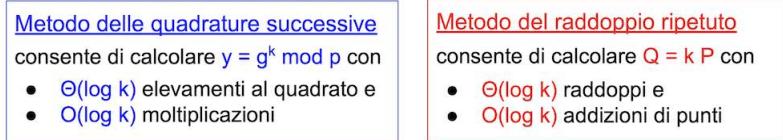
Per definire un sistema crittografico a chiave pubblica usando le curve ellittiche occorre individuare una funzione one-way trap-door che ne garantisca la sicurezza.

La chiave sta nel fatto che l'operazione di **addizione di punti** di una curva ellittica su un campo finito presenta analogie con l'operazione di prodotto modulare. Di conseguenza, per le curve ellittiche si definisce una funzione one-way trap-door analoga all'elevamento a potenza/logaritmo discreto nell'algebra modulare.

In particolare, fissato un intero  $k$  si ha:



Infatti, entrambe le operazioni si possono eseguire in tempo polinomiale:



in cui il **metodo del raddoppio ripetuto** consiste nei seguenti passi:

1. Si scrive anzitutto  $k$  come somma di potenze di 2

$$k = \sum_{i=0}^t k_i 2^i$$

ove  $k_t \dots k_0$  è la rappresentazione binaria di  $k$ , e  $t = \lfloor \log_2 k \rfloor$ .

2. Si calcolano in successione i punti  $2P, 4P, \dots, 2^t P$ , ottenendo ciascuno dal raddoppio del precedente.
3. Si calcola infine  $Q$  come somma dei punti  $2^i P$ , con  $i \in \{0, \dots, t\}$  e tale che  $k_i = 1$ .

**Esempio:**  $Q = 13P = (8 + 4 + 1)P$  si calcola con 3 raddoppi:

$$2P \quad 4P = 2(2P) \quad 8P = 2(4P)$$

e due addizioni:

$$Q = P + 4P + 8P$$

### 18.2.1 Logaritmo discreto per le curve ellittiche

Dati due punti  $P$  e  $Q$ , si pone il problema di trovare, se esiste, il più piccolo intero  $k$  tale che  $Q = kP$ . In tale contesto,  $k$  è chiamato *logaritmo in base  $P$  del punto  $Q$* .

**Il problema del logaritmo discreto per le curve ellittiche risulta computazionalmente difficile.** Infatti, tutti gli algoritmi noti hanno complessità esponenziale se i parametri della curva sono scelti bene. Addirittura, non è stato ancora trovato un algoritmo per risolvere questo problema che migliori il metodo a forza bruta, basato sul calcolo di tutti i multipli di  $P$  fino a trovare  $Q$ , improponibile se  $k$  è sufficientemente grande.

Ed eccoci così trovata la nostra funzione one-way trap-door!

Funzione **computazionalmente trattabile** in una direzione  
 calcolo di  $Q = kP$  dati  $P$  e  $k$   
 ma praticamente **intrattabile** nell'altra  
 calcolo di  $k$  dati  $P$  e  $Q = kP$

**Osservazione:** È immediato notare l'analogia con il calcolo del logaritmo discreto.

## 18.3 Applicazioni in crittografia

Vediamo adesso due algoritmi crittografici che usano le curve ellittiche. Da notare che essi non sono altro che riproposizioni di algoritmi classici basati sul problema del logaritmo discreto, ottenute sostituendo le operazioni dell'algebra modulare con le operazioni sui punti di una curva prima o binaria.

In tale contesto, abbiamo i seguenti cambiamenti:

- prodotto → somma di punti
- esponenziazione → moltiplicazione scalare di un intero per un punto della curva

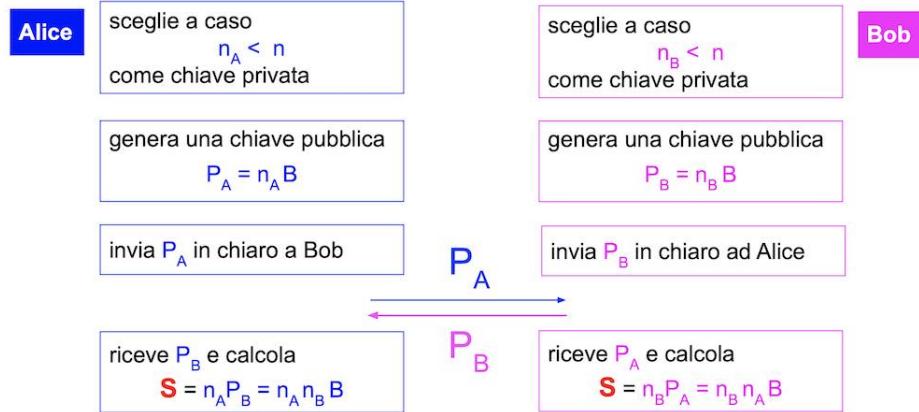
### 18.3.1 DH su curve ellittiche

Variante più sicura del DH classico, prevede il seguente funzionamento:

- Alice e Bob si accordano pubblicamente su un **campo finito** e su una **curva ellittica** definita su tale campo
- Quindi scelgono, nella curva, un punto  $B$  di **ordine molto grande**

**Osservazione:** L'ordine di un punto  $B$  della curva è il più piccolo intero positivo  $n$  tale che  $nB = O$ .

A questo punto, il funzionamento è strettamente analogo a quello della versione classica del protocollo:



Alice e Bob condividono lo stesso punto  $S$  determinato dalle scelte casuali di entrambi. A questo punto, per trasformare  $S$  in una chiave segreta  $K$  per la cifratura simmetrica convenzionale, occorre **convertirlo** in un numero intero. Per esempio, si può porre:

$$K = x_S \bmod 2^{256}$$

cioè  $K$  uguale agli ultimi 256 bit dell'ascissa di  $S$ .

### Crittoanalisi

Il crittoanalista **passivo** intercetta i messaggi  $P_A$  e  $P_B$  scambiati in chiaro. Egli conosce anche i parametri della curva ellittica impiegata e il punto  $B$ , che conosce.

A questo punto, per calcolare  $S$  dovrebbe calcolare  $n_A$ , dati  $B$  e  $P_A$ , oppure  $n_B$ , dati  $B$  e  $P_B$ . Egli dovrebbe quindi risolvere il problema del logaritmo discreto per le curve ellittiche, certamente **intrattabile** date le dimensioni dei valori coinvolti.

**Osservazione:** Il protocollo è comunque vulnerabile agli attacchi attivi di tipo **man-in-the-middle**, al pari della versione basata sull'algebra modulare.

### 18.3.2 ElGamal su curve ellittiche

Il cifrario di ElGamal su curve ellittiche prevede un diverso modo di creazione delle chiavi di cifratura(decifrazione) del messaggio(crittogramma).

#### Creazione delle chiavi

1. Si sceglie un numero primo  $p$ , una curva ellittica prima  $E_p(a, b)$  e un punto  $B$  della curva, di ordine  $n$  elevato.
2. Ogni utente  $U$  genera la propria coppia chiave pubblica/chiave privata scegliendo a caso un intero  $n_U < n$  e calcolando  $P_U = n_U B$ , ottenendo quindi:
  - $k_{pub} = P_U$
  - $k_{priv} = n_U$

### Cifratura e decifrazione

Per cifrare un messaggio  $m$  codificato come numero intero utilizzando le curve ellittiche, occorre per prima cosa **trasformare  $m$  in un punto di una curva**.

Tuttavia, non è noto alcun algoritmo deterministico polinomiale per effettuare tale trasformazione. Esistono però algoritmi **randomizzati** molto efficienti che hanno una probabilità arbitrariamente bassa di fallire, cioè di produrre un punto **non** della curva.

Vediamo, tra i possibili algoritmi randomizzati, l'**algoritmo di Koblitz**:

Dato  $m < p$ , trovare  $P_m \in E_p(a, b)$ .

Usando  $m$  come ascissa, la probabilità di trovare un punto della curva è pari alla probabilità che  $(m^3 + am + b) \bmod p$  sia un residuo quadratico, cioè circa  $\frac{1}{2}$ .

Si fissa un intero positivo  $h$  tale che  $(m+1)h < p$ , quindi si esegue l'algoritmo:

```
For i = 0 to h-1 {
    x = m * h + i
    se esiste la radice quadrata y di  $x^3 + ax + b \pmod{p}$ 
        return Pm = (x, y)
    }
return failure
```

La probabilità che, per ogni valore di  $x$  considerato,  $(x^3 + ax + b) \bmod p$  non sia un residuo quadratico, è circa  $\frac{1}{2}$ . Quindi, l'algoritmo ha una probabilità complessiva di successo pari a circa  $1 - \frac{1}{2^h}$ , dato che si itera al più  $h$  volte.

Per risalire a  $m$  dato  $P_m = (x, y)$ , basta calcolare  $m = \lfloor x/h \rfloor$

Avendo trovate il modo di trasformare  $m$  nel punto  $P_m$ , le funzioni di cifratura e decifrazione diventano immediate:

- **CIFRATURA** (Alice)

1. Trasforma il messaggio  $m$  in un punto  $P_m$  sulla curva.
2. Sceglie a caso un intero  $r < n$ .
3. Calcola la coppia di crittogrammi (punti sulla curva)

$$V = rB \quad W = P_m + rP_{Bob}$$

4. Invia la coppia di punti  $(V, W)$  a Bob.

- **DECIFRAZIONE** (Bob)

1. Riceve la coppia di punti  $(V, W)$ .
2. Ricostruisce il punto  $P_m$  utilizzando la sua chiave privata  $n_{Bob}$

$$P_m = W - n_{Bob}V$$

infatti:

$$W - n_{Bob}V = P_m + rP_{Bob} - n_{Bob}V = P_m + rn_{Bob}B - n_{Bob}rB = P_m$$

3. Trasforma  $P_m$  nel messaggio  $m$ .

### Crittoanalisi

Per risalire a  $m$ , un crittoanalista che ha intercettato il crittogramma  $(V, W)$ , dovrebbe fare **una** delle due seguenti cose:

- Calcolare  $r$  da  $B$  e  $V$
- Calcolare la chiave privata  $n_{Bob}$  da  $B$  e  $P_{Bob}$

In ogni caso, dovrebbe comunque risolvere il problema del logaritmo discreto per le curve ellittiche!

## 18.4 Sicurezza della crittografia su curve ellittiche

La sicurezza della crittografia su curve ellittiche è legata, come ormai sappiamo, alla difficoltà di calcolare il logaritmo discreto di un punto, problema per cui non è noto alcun algoritmo efficiente di risoluzione. Tale problema è addirittura molto più difficile dei tradizionali problemi della fattorizzazione degli interi e del logaritmo discreto nell'algebra modulare! Infatti, per quest'ultimi esiste un algoritmo subesponenziale (**index calculus**) di risoluzione, motivo per cui algoritmi crittografici basati su di essi richiedono chiavi con migliaia di bit.

L'*index calculus* frutta una struttura algebrica dei campi finiti che **non** è presente sulle curve ellittiche. Quindi non può essere applicato ad esse. Di fatto, ad oggi nessuno è stato capace di progettare algoritmi analoghi ad *index calculus* per il problema del logaritmo discreto per le curve ellittiche. Quindi i protocolli per tali curve sembrano invulnerabili a questo tipo di attacchi.

Il migliore attacco noto (**Pollard  $\rho$** ) richiede in media  $O(2b/2)$  operazioni per chiavi di  $b$  bit. Di conseguenza, tale attacco richiede **tempo esponenziale**. Questo spiega inoltre perché i cifrari su curve ellittiche richiedono il doppio di bit rispetto all'AES per garantire un equivalente livello di sicurezza.

TDEA, AES (bit della chiave)	RSA e DH (bit del modulo)	ECC (bit dell'ordine)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

**Osservazione:** Esistono attacchi efficaci contro alcune famiglie di curve, ma queste famiglie sono note e facilmente evitate.

# Chapter 19

## Lezione 19 (24/11/2022) Identificazione & autenticazione

Finora abbiamo trattato la crittografia solo come un mezzo per garantire la **confidenzialità**. Tuttavia, gli algoritmi crittografici permettono anche di garantire sia **identificazione** che **autenticazione**. In particolare, definiamo:

- **Identificazione:** Accertare l'**identità di un utente** che svolge azioni all'interno di un certo sistema (e.g. inviare un messaggio, accedere un servizio).
- **Autenticazione:** Accertare l'**integrità di un messaggio** inviato da un utente identificato.

**Osservazione:** Notiamo subito come l'autenticazione sia una proprietà più forte dell'identificazione.

Sia ben chiaro che queste due funzionalità, così come la firma digitale che vediamo adesso, hanno lo scopo di contrastare gli **attacchi attivi** ad un sistema crittografico. Si pensi infatti agli attacchi di tipo man-in-the-middle; se l'autenticazione è garantita, allora il sistema diventa virtualmente immune a tali attacchi.

### Firma digitale

A partire dai concetti di identificazione ed autenticazione sopra visti, nasce il concetto di **firma digitale**.

La firma digitale è uno strumento crittografico che conferisce le seguenti proprietà al messaggio sul quale è applicato:

- Il mittente **non** deve poter negare di aver inviato un messaggio  $m$ .
- Il destinatario deve essere in grado di **autenticare**  $m$ .
- Il destinatario **non** deve poter sostituire che  $m' \neq m$  sia il messaggio inviato dal mittente.

Inoltre, **tutto deve essere verificabile da una terza parte**.

**Osservazione:** Identificazione, autenticazione e firma digitale non sono indipendenti, bensì ognuna estende le caratteristiche dall'altra:

- L'autenticazione di un messaggio garantisce l'identificazione del mittente.
- L'apposizione della firma digitale garantisce l'autenticazione del messaggio.

**Nota:** Lo strumento di firma digitale può essere realizzato mediante algoritmi basati su cifrari asimmetrici o simmetrici. Noi vedremo quelli basati su cifrari **asimmetrici**.

## 19.1 Funzioni hash

Una funzione hash  $H : X \rightarrow Y$  è una funzione tale che

$$n = |X| \gg m = |Y|$$

Questo implica inesorabilmente che

$$\exists X_1, X_2, \dots, X_m \subseteq X \text{ disgiunti} : X = X_1 \cup \dots \cup X_m \wedge \forall i . \forall x \in X_i . H(x) = y_i$$

**Osservazione:** Tante formule per dire una cosa ovvia, ma vabbè.

### 19.1.1 Proprietà

Una **buona** funzione hash deve assicurare che:

- i sottoinsiemi  $X_1, \dots, X_m$  abbiano circa la **stessa cardinalità**: due elementi estratti a caso da  $X$  hanno probabilità circa  $\frac{1}{m}$  di avere la stessa immagine in  $Y$ .
- elementi di  $X$  molto *simili* tra loro appartengano a due **sottoinsiemi diversi**: se  $X$  è un insieme di interi, due elementi con valori prossimi devono avere immagini diverse.
- vi sia un'adeguata **gestione delle collisioni**: l'algoritmo che impiega la funzione hash dovrà affrontare la situazione in cui più elementi di  $X$  hanno la stessa immagine in  $Y$ .

Inoltre, se la funzione è applicata in **crittografia** (come nel nostro caso) deve (dovrebbe...) rispettare anche le seguenti proprietà:

- **Grandezza dell'input variabile:**  $H$  può essere applicata ad un blocco di dati di qualsiasi grandezza.
- **Grandezza dell'output fissata:**  $H$  produce un output di lunghezza predefinita.
- **Efficienza:**  $y = H(x)$  è computazionalmente **facile** da calcolare per ogni  $x \in X$ .
- **Proprietà one-way:** per praticamente ogni  $y \in Y$  è computazionalmente **difficile** determinare **un**  $x \in X$  tale che  $H(x) = y$ .
- **Collision-resistance:** È computazionalmente **difficile** trovare una coppia  $(x_1, x_2)$  tale che  $H(x_1) = H(x_2)$ .
- **Pseudorandomicità:** L'output di  $H$  soddisfa i test standard di pseudorandomicità.

Vediamo a questo punto alcune delle funzioni hash usate in crittografia.

### 19.1.2 MD5

**MD (Message Digest)** consiste in una famiglia di algoritmi: il primo non fu mai pubblicato, mentre invece furono resi pubblici MD2 ed il successivo MD4. In entrambi questi algoritmi furono trovate gravi debolezze, cosicché Rivest propose nel 1992 una nuova versione, **MD5**.

MD5 riceve in input una sequenza  $S$  di **512 bit**, e produce un'immagine di **128 bit**, quindi  $S$  è *digerita* riducendola ad un quarto.

## Problemi

È stato dimostrato che MD5 non resiste alle collisioni, e nel 2004 si sono individuate serie debolezze. Al giorno d'oggi, MD5 è considerato **obsoleto**.

### 19.1.3 RIPEMD-160

RIPEMD-160, come suggerisce il nome, è un'evoluzione degli algoritmi MD. Esso produce immagini di **160 bit** ed è privo dei difetti di MD5.

## 19.2 SHA

**SHA (Secure Hash Algorithm)** è una famiglia di algoritmi progettata da NIST ed NSA nel 1993. Tali algoritmi vengono adottati quando al resistenza alle collisioni è cruciale per il sistema, e nella pratica tale famiglia è diventata uno standard al giorno d'oggi.

Gli algoritmi SHA operano su sequenze lunghe fino a  $2^{64}$  **bit**, e producono immagini di **160 bit**. Inoltre, gli algoritmi sono **crittograficamente sicuri**, in quanto rispettano i requisiti delle funzioni one-way e generano immagini molto diverse per sequenze molto simili.

### Storico

- **SHA-0 (1993)**: proposta dal NIST, presto ritirata a causa di una debolezza interna scoperta dall'NSA.
- **SHA-1 (1995)**: progettato da NSA, uso raccomandato dal NIST.
- **SHA-2 (2001)**: quattro funzioni della famiglia SHA, progettate da NSA e pubblicate dal NIST, caratterizzate da **digest più lunghi**.
- **SHA-3 (2012)**: Nel 2007, il NIST ha sollecitato proposte per un nuovo algoritmo hash, a causa della presenza di attacchi terorici su SHA-1. Nel 2012, si è conclusa la valutazione, ed è stato selezionato una funzione hash, di progettazione non governativa (**algoritmo Keccak**). Rilasciato ufficialmente nel 2015.

### 19.2.1 SHA-1

SHA-1 opera su sequenze lunghe fino a  $2^{64} - 1$  **bit**, e produce immagini di **160 bit**.

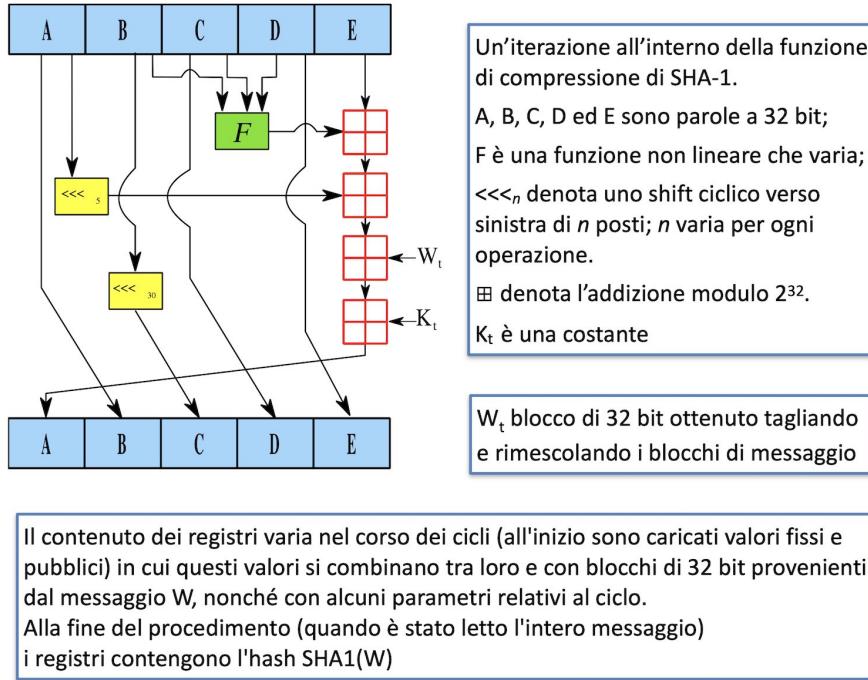
**Osservazione:** Al giorno d'oggi, SHA-1 non è più certificata come standard. In ogni caso, tutte le versioni successive hanno struttura simile a SHA-1, quindi vale la pena di studiarla.

### Funzionamento

SHA-1 opera su **blocchi di 160 bit**, contenuti in un buffer di **5 registri** di 32 bit ciascuno, all'interno dei quali sono caricati inizialmente dei valori **pubblici**. Quindi, il messaggio  $m$  viene concatenato con una **sequenza di padding**, che ne rende la lunghezza un multiplo di 512 bit.

Il contenuto dei vari registri varia nel corso dei cicli successivi, in cui questi valori si combinano tra loro e con blocchi di 32 bit provenienti da  $m$ .

Alla fine del procedimento, i registri contengono  $\text{SHA-1}(m)$ .



## 19.3 Identificazione

Passiamo adesso a vedere come si può effettuare l'**identificazione** mediante l'uso di funzioni hash.

In particolare, vediamo come può essere effettuata l'identificazione sia su **canali sicuri** che su **canali insicuri**. In tale contesto, viene definito sicuro un canale all'interno del quale la comunicazione è **confidenziale** e non può essere alterata o manomessa da terzi.

### 19.3.1 Identificazione su canali sicuri

Vediamo il caso in cui un certo utente debba accedere al suo account su un computer **UNIX**.

In tale contesto, l'utente inizia il collegamento inviando **in chiaro** (canale sicuro) username e password. Il meccanismo di identificazione prevede una **cifratura delle password**, realizzata con funzioni hash one-way.

Quando un utente  $U$  fornisce per la prima volta una password  $P$ , il sistema associa a  $U$  **due sequenze binarie** (che memorizza nel file delle password al posto di  $P$ ):

- **S:** seme prodotto da un generatore pseudocasuale.
- **Q = H(P:S):** con  $H$  una funzione hash one-way.

Ad ogni successiva connessione di  $U$ , il sistema:

1. recupera  $S$  dal file delle password
2. concatena  $S$  alla password  $P$  fornita da  $U$
3. calcola l'immagine one-way della nuova sequenza ( $H(P : S)$ )
4. se  $H(P : S) = Q$ , allora l'identificazione ha successo

**Osservazione:** Si noti che in questo modo il computer **non** conosce direttamente la password dell'utente, ed **un accesso illecito al file delle password non fornisce informazioni interessanti**. Infatti, è computazionalmente difficile ricavare la password originale dalla sua immagine one-way.

### 19.3.2 Identificazione su canali insicuri

Qui la situazione non è semplice come prima, dato che un utente non può inviare le proprie credenziali in chiaro e sperare che non gliele fottano. Infatti, se il canale è insicuro, la password può essere intercettata durante la sua trasmissione in chiaro. Perciò, il sistema non dovrebbe mai maneggiare direttamente la password, ma solamente una sua immagine inattaccabile.

A tale scopo, si sfrutta la crittografia asimmetrica, solo nel modo contrario a come l'abbiamo precedentemente usata per garantire confidenzialità: supponiamo che  $(e, n)$ ,  $(d)$  siano chiave pubblica e privata di un utente  $U$  che richiede l'accesso ai servizi offerti dal sistema  $S$ , allora:

1.  $S$  genera un numero **casuale**  $r < n$  e lo invia ad  $U$ .
2.  $U$  calcola
 
$$f = r^d \text{ mod } n$$
 con la sua chiave **privata**. Quindi spedisce  $f$  ad  $S$ .
3.  $S$  verifica la correttezza del valore ricevuto verificando se
 
$$f^e \text{ mod } n = r$$

Se ciò avviene, l'identificazione ha successo.

In questo modo, l'identificazione è garantita, dato che  $f$  può essere correttamente generata solo da  $U$ , che è l'unico (si spera) a possedere  $d$ .

**Osservazione:** Si noti come le operazioni di cifratura e decifrazione sono invertite rispetto all'impiego standard nell'RSA. Ciò è reso possibile dal fatto che tale operazioni sono **commutative**.

### Problemi

Il problema nell'usare questo metodo di identificazione, è che  $S$  chiede a  $U$  di applicare la sua chiave privata a una sequenza  $r$  che  $S$  stesso ha generato. Se  $S$  è malevolo, potrebbe aver scelto  $r$  di proposito, e non casualmente, per ricavare qualche informazione sulla chiave privata di  $U$ .

Per questo motivo, esiste un altro protocollo di identificazione, detto a *conoscenza zero*, il quale impedisce che da una comunicazione si possa estrarre più di quanto sia nelle intenzioni del comunicatore.

## 19.4 Autenticazione

Va bene definire dei protocolli di identificazione, ma sui canali insicuri si pone il problema di sapere se un messaggio è stato corrotto o meno. Insomma, il destinatario deve autenticare il messaggio accertando l'identità del mittente **e l'integrità di  $m$** .

A tal fine:

1. Mittente e destinatario concordano una **chiave segreta**  $k$ , diversa dalla eventuale chiave di sessione.
2. Il mittente allega al messaggio un **MAC (Message Authentication Code)**  $A(m, k)$ , allo scopo di garantire la provenienza e l'integrità del messaggio.
3. Il mittente spedisce la coppia
  - in chiaro:  $\langle m, A(m, k) \rangle$
  - oppure cifrata:  $\langle C(c, k_{session}), A(m, k) \rangle$
 a seconda del fatto che voglia anche mantenere la confidenzialità o meno.
4. Il destinatario entra in possesso di  $m$ , dopo averlo eventualmente decifrato.
5. Essendo a conoscenza di  $A$  e  $k$ , il destinatario calcola  $A(m, k)$ , e confronta il valore ottenuto con quello inviato dal mittente, per verificare che il MAC ricevuto corrisponda al messaggio a cui risulta allegato.
6. Se la verifica ha successo, il messaggio è autenticato, altrimenti viene scartato.

#### 19.4.1 MAC

Il MAC è un'immagine breve del messaggio, che può essere stata generata solo da un mittente conosciuto dal destinatario. Ne sono state proposte varie realizzazioni, basate su cifrari asimmetrici, simmetrici e sulle funzioni hash one-way.

Nel caso di MAC realizzato mediante funzioni hash one-way, si ha

$$A(m, k) = H(m : k)$$

In tale contesto, risulta computazionalmente difficile per un crittoanalista passivo scoprire la chiave segreta  $k$ , dato che per ricavarla bisognerebbe invertire  $H$ .

Inoltre, un crittoanalista attivo non può sostituire facilmente il messaggio  $m$  con un altro messaggio  $m'$ , visto che dovrebbe allegare alla comunicazione di  $m'$  il MAC  $A(m', k)$ , che può produrre solo conoscendo  $k$ .

**Osservazione:** Usando un cifrario a blocchi in modalità CBC, si può usare il blocco finale del crittogramma come MAC. Il blocco finale è infatti funzione dell'intero messaggio.

# Chapter 20

## Lezione 20 (28/11/2022)

### 20.1 Firma digitale

Avendo visto protocolli di identificazione ed autenticazione, ci concentriamo adesso sui protocolli di **firma digitale**, che come abbiamo detto estendono quelli di autenticazione, e quindi anche di identificazione. Infatti, in modo molto brutto, possiamo dire che

$$\text{firmaDigitale} > \text{autenticazione} > \text{identificazione}$$

#### 20.1.1 Proprietà

Una firma digitale, al pari di una manuale:

- **è autentica e non falsificabile:** prova che chi l'ha prodotta è chi ha sottoscritto il documento.
- **non è riutilizzabile:** è legata strettamente al documento su cui è stata apposta.
- **rende inalterabile il documento su cui è apposta:** chi ha prodotto la firma è sicuro che questa si riferirà solo al documento sottoscritto nella sua forma originale.
- **non può essere ripudiata da chi l'ha apposta:** costituisce prova legale di un accordo o dichiarazione.

Tuttavia, la firma digitale non può semplicemente consistere in una digitalizzazione di un documento firmato manualmente (per ovvi motivi). Infatti, la firma digitale deve **dipendere** dal documento sul quale è apposta.

Come vedremo, la firma digitale verrà implementata come una funzione matematica che prende in input proprio il documento da firmare.

**Nota:** Noi vedremo protocolli di firma digitale basati su cifrari **asimmetrici**, anche se esistono altri su cifrari simmetrici.

Vediamo in particolare 4 protocolli di firma digitale, anche se i primi due sono ormai obsoleti.

#### 20.1.2 Protocollo 1

Il primo protocollo fu ideato da Diffie e Hellman nel 1976, e prevede l'invio di un messaggio  $m$  in **chiaro e firmato**.

### Funzionamento

Consideriamo un utente  $U$  con chiavi  $k_U[prv]$  e  $k_U[pub]$ . Quindi consideriamo  $C$  e  $D$ , due funzioni di cifratura e decifrazione di un cifrario asimmetrico.

1.  $U$  genera la **firma**

$$f = D(m, k_U[prv])$$

per  $m$ .

2.  $U$  spedisce a  $V$  la tripla  $(U, m, f)$ .
3.  $V$  riceve la tripla e verifica l'autenticità della firma controllando che

$$C(f, k_U[pub]) = m$$

**Osservazione:** I processi di firma e verifica impiegano le funzioni  $C$  e  $D$  in ordine inverso a quello standard,  $C$  e  $D$  devono quindi essere **commutative**.

### Proprietà

Questo protocollo soddisfa tutti i requisiti della firma manuale, visti prima. Infatti:

- $f$  è autentica e non falsificabile, dato che viene prodotta sfruttando la chiave privata di  $U$ , nota solo a lui. Inoltre, ricavare la chiave privata dalla firma (invertendo  $D$ ) è un problema intrattabile, visto che  $D$  è one-way.
- $f$  non è riutilizzabile su un altro documento  $m' \neq m$ , poiché  $f$  è funzione di  $m$ .
- $m$  non può essere alterato se non da  $U$ , altrimenti la firma  $f$  diventa inconsistente.
- poiché solo  $U$  può aver prodotto la firma, egli non può ripudiarla.

### Problemi

Problema da considerare è che la tripla ha praticamente lunghezza doppia del messaggio originale, dato che  $f$  è circa lunga quanto  $m$ .

In ogni caso, il problema essenziale di questo protocollo è che il messaggio viene inviato in chiaro, quindi non è garantita la **confidenzialità**. Inoltre, anche se il messaggio venisse cifrato, comunque chiunque lo potrebbe ricavare in tempo polinomiale dalla verifica della firma.

**Considerazioni personali:** Se si cifrasse l'intera tripla, magari con una chiave di sessione, si potrebbe facilmente garantire la confidenzialità.

#### 20.1.3 Protocollo 2

Questo secondo protocollo prevede l'invio di un messaggio  $m$  **cifrato e firmato**.

### Funzionamento

1.  $U$  genera la firma

$$f = D(m, k_U[prv])$$

per  $m$ .

2. Calcola il **crittogramma firmato**

$$c = C(f, k_V[pub])$$

con la chiave pubblica di  $V$  (destinatario).

3. Spedisce la coppia  $(U, c)$  a  $V$ .
4.  $V$  riceve tale coppia e decifra il crittogramma:

$$D(c, k_V[prv]) = f$$

5. Ottiene  $m$  cifrando  $f$  con la chiave pubblica di  $U$ :

$$C(f, k_U[pub]) = C(D(m, k_U[prv]), k_U[pub]) = m$$

6. Se  $m$  è significativo, allora lo accetta. In caso contrario lo scarta, dato che vuol dire che la firma è stata compromessa. Infatti, un utente diverso da  $U$  ha probabilità praticamente nulla di generare un crittogramma di significato accettabile se cifrato con la chiave pubblica di  $U$ .

### Algoritmo

Tale protocollo sfrutta il cifrario RSA. Di conseguenza abbiamo:

- $k_U[prv] = d_U$
- $k_U[pub] = (e_U, n_U)$
- $k_V[prv] = d_V$
- $k_V[pub] = (e_V, n_V)$

e quindi:

1.  $U$  genera la firma  $f = m^{d_U} \bmod n_U$  per  $m$ .
2. Cifra  $f$  con la chiave pubblica di  $V$ :  $c = f^{e_V} \bmod n_V$ .
3. Spedisce a  $V$  la coppia  $(U, c)$ .
4.  $V$  riceve tale coppia e decifra  $c$ :  $c^{d_V} \bmod n_V = f$ , quindi ottiene  $m$  con la chiave pubblica di  $U$ :  $f^{e_U} \bmod n_U = m$ .
5. Se  $m$  è significativo, allora conclude che è autentico.

## Vincoli

Per la correttezza del procedimento, è necessario che  $n_U \leq n_V$  perché risulti  $f < n_V$  e possa quindi essere cifrata correttamente.

Tuttavia, vale il vincolo al contrario se anche  $V$  vuole inviare un messaggio dirmato ad  $U$ . Quindi, si fa che ogni utente stabilisce **chiavi distinte** per la firma e per la cifratura:

- Si fissa **pubblicamente**  $H$  molto grande.
- Si sceglie una **chiave di firma**  $< H$ .
- Si sceglie una **chiave di cifratura**  $> H$ .

## Attacchi

Anche se il valore elevato di  $H$  previene attacchi esaurienti, in ogni caso il protocollo è soggetto a vari possibili attacchi basati sulla possibilità che un crittoanalista si procuri la firma di un utente su messaggi apparentemente privi di senso.

In tale contesto, vediamo uno dei possibili tipi di attacco:

Supponiamo che  $V$  invii una risposta automatica (**ack**) ad  $U$  ogni volta che riceve un suo messaggio  $m$ , e che l'ack sia costituito dal crittogramma della firma di  $V$  su  $m$ .

In tale contesto, un crittoanalista attivo  $X$  può decifrare i messaggi inviati a  $V$  nel seguente modo:

1.  $U$  invia il crittogramma  $c$  a  $V$ :

$$c = C(f, k_V[\text{pub}]) \quad f = D(m, k_U[\text{prv}])$$

2.  $X$  intercetta  $c$ , lo rimuove dal canale e lo rispedisce a  $V$ . Quindi  $V$  crede che  $c$  sia stato inviato da  $X$ .

3.  $V$  decifra  $c$ :

$$f = D(c, k_V[\text{prv}])$$

Quindi verifica la firma con la chiave pubblica di  $X$ , ottenendo:

$$m' = C(f, k_X[\text{pub}])$$

4.  $m' \neq m$  è privo di significato, ma  $V$  invia comunque l'ack  $c'$  a  $X$ :

$$f' = D(m', k_V[\text{prv}]) \quad c' = C(f', k_X[\text{pub}])$$

5.  $X$  decifra  $c'$  trovando  $f'$ :

$$D(c', k_X[\text{prv}]) = D(C(f', k_X[\text{pub}]), k_X[\text{prv}]) = f'$$

6. Verifica  $f'$  trovando  $m'$ :

$$C(f', k_V[\text{pub}]) = C(D(m', k_V[\text{prv}]), k_V[\text{pub}]) = m'$$

7. Ricava  $f$  da  $m'$ :

$$D(m', k_X[\text{prv}]) = D(C(f, k_X[\text{pub}]), k_X[\text{prv}]) = f$$

8. Verifica  $f$  con la chiave pubblica di  $U$  trovando  $m$

$$C(f, k_U[\text{pub}]) = C(D(m, k_U[\text{prv}]), k_U[\text{pub}]) = m$$

**Osservazione:** L'attacco sopra esposto può essere evitato semplicemente bloccando l'ack automatico da parte di  $V$ . Infatti,  $V$  dovrebbe inviare l'ack solo dopo un esame preventivo di  $m$ , scartando quindi i messaggi non significativi.

#### 20.1.4 Protocollo 3

Questi ultimi due protocolli che vediamo prevengono attacchi come quello visto sopra, in quanto essi **evitano la firma diretta del messaggio**. In questo modo, se un crittoanalista attivo ricava la firma  $f$  del messaggio inviato e la verifica, tutto ciò che ottiene sono dati non significativi.

Infatti, la firma digitale viene apposta su una **immagine del messaggio** ottenuta con una **funzione hash one-way pubblica** (e.g. MD5, SHA). Ciò garantisce la verifica della firma restituiscia appunto l'immagine del messaggio, dalla quale è intrattabile risalire al messaggio originale.

#### Funzionamento

1.  $U$  calcola  $H(m)$  e genera la firma:

$$f = D(H(m), k_U[prv])$$

2. Calcola separatamente il crittogramma del messaggio:

$$c = C(m, k_V[pub])$$

3. Spedisce a  $V$  la tripla  $(U, c, f)$
4.  $V$  riceve tale tripla e decifra il crittogramma  $c$ :

$$m = D(c, k_V[prv])$$

5. Ottenuto  $m$ , calcola  $H(m)$  e  $C(f, k_U[pub])$  (verifica la firma).
6. Se i due valori ottenuti corrispondono, allora conclude che il messaggio è autentico, altrimenti lo scarta.

## 20.2 Certificato digitale

Prima di proseguire con il quarto protocollo, bisogna introdurre il concetto di **certificato digitale**.

Problema comune a tutti i protocolli visti finora, compreso il terzo, è che le **chiavi di cifratura sono pubbliche e non richiedono un incontro diretto tra gli utenti per il loro scambio**.

Adesso, ragionando un attimo, in tale contesto cosa vieterebbe ad un crittoanalista attivo di fare un attacco man-in-the-middle durante la fase di recupero delle chiavi pubbliche?

**Osservazione:** Il terzo protocollo visto è immediatamente violato se il crittoanalista effettua un man-in-the-middle fornendo ad  $U$  la **sua** chiave pubblica, spacciandola per quella di  $V$ .

In particolare, il crittoanalista  $X$  devia le comunicazioni che provengono da  $U$  e  $V$  e le dirige verso se stesso. Di conseguenza:

1.  $U$  richiede a  $V$  la sua chiave pubblica.

2.  $X$  intercetta la risposta contenente  $k_V[pub]$  e la sostituisce con la sua chiave pubblica  $k_X[pub]$ .
3.  $X$  si pone in ascolto in attesa dei crittogrammi spediti da  $U$  a  $V$ , cifrati mediante la chiave pubblica di  $X$ , anche se  $U$  ne è ignaro.
4.  $X$  rimuove dal canale ciascuno di questi crittogrammi, e lo decrittua risalendo al messaggio originale, che cifra mediante  $k_V[pub]$  rispedendolo a  $V$ .
5.  $U$  e  $V$  non si accorgono della presenza di  $X$  se il processo di intercettazione e rispedizione è sufficientemente veloce da rendere il relativo ritardo apparentemente attribuibile alla rete.

### 20.2.1 Certification Authority

Insomma, si è capito che lo scambio delle chiavi è un passo **cruciale** che non deve in alcun modo essere compromesso, pena la totale violazione di qualunque protocollo di sicurezza utilizzato per garantire confidenzialità.

Per questo motivo sono nate la **certification authorities (CA)**. Esse sono infrastrutture che **garantiscono la validità delle chiavi pubbliche** e ne regolano l'uso, gestendone la distribuzione sicura alle due entità che vogliono comunicare.

A tal fine, la CA autentica l'associazione (**utente, chiave pubblica**) emettendo un **certificato digitale**, composto dalla chiave stessa e da una lista di informazioni relative al suo proprietario, **opportunamente firmate dalla CA**.

Per svolgere correttamente il suo ruolo, la CA mantiene un **archivio di chiavi pubbliche sicuro**, accessibile in lettura a tutti ed in scrittura solo ai pochi che sono autorizzati.

La chiave pubblica della CA è **nota** agli utenti, i quali devono mantenerla protetta da attacchi esterni e che utilizzano per verificare la firma della CA stessa sui certificati delle chiavi richieste.

### 20.2.2 Certificato digitale

Vediamo quindi in maggior dettaglio in cosa consiste un certificato digitale rilasciato dalla CA. Esso contiene:

- **numero di versione**
- **nome della CA** che lo ha rilasciato
- **numero seriale**, che lo individua univocamente all'interno della CA emittente
- **specifica dell'algoritmo usato** dalla CA per creare la firma elettronica
- **periodo di validità** del certificato
- **nome dell'utente** a cui questo certificato si riferisce e una serie di informazioni a lui legate
- **indicazione del protocollo a chiave pubblica adottato** da questo utente per la cifratura e la firma: nome dell'algoritmo, suoi parametri, e **chiave pubblica dell'utente**
- **firma della CA eseguita su tutte le informazioni precedenti**

Quindi, un utente  $U$  che vuole comunicare con  $V$  farà la seguenti cose:

1.  $U$  richiede  $k_V[pub]$  alla CA, la quale risponde inviando ad  $U$  il certificato digitale  $cert_V$  di  $V$ .

2. Poiché  $U$  conosce  $k_{CA}[pub]$ , può controllare l'autenticità del certificato verificandone il periodo di validità e la firma prodotta dalla CA su di esso.
3. Se tutti i controlli vanno a buon fine,  $k_V[pub]$  nel certificato è corretta e  $U$  avvia la comunicazione con  $V$ .

Si noti che un crittoanalista potrebbe intromettersi solo falsificando la certificazione, ma si assume che la CA sia fidata e il suo archivio di chiavi sia inattaccabile.

### Organizzazione del CA

Esistono in ogni stato diverse CA organizzate ad albero. In tale contesto, la verifica di un certificato risulta più complicata, e si svolge attraverso una catena di verifiche che vanno dalla CA di  $U$  alla CA di  $V$ .

Ogni utente  $U$  mantiene localmente al sistema una copia del proprio certificato  $cert_U$  e una copia di  $k_{CA}[pub]$ . In questo modo,  $U$  interagisce con la CA una sola volta (per la durata di validità del certificato), e poi la gestione delle chiavi pubbliche diventa **decentralizzata**.

#### 20.2.3 Protocollo 4

Il quarto protocollo lo vediamo solo adesso perché prevedibilmente sfrutta i certificati digitali delle chiavi pubbliche al fine di evitare gli attacchi di cui abbiamo parlato prima.

#### Funzionamento

1.  $U$  si procura il certificato  $cert_V$  di  $V$ .
2. Calcola  $H(m)$  e genera la firma:

$$f = D(H(m), k_U[prv])$$

3. Calcola il crittogramma:

$$c = C(m, k_V[pub])$$

4. Spedisce a  $V$  la tripla  $(cert_U, c, f)$ .
5.  $V$  riceve tale tripla e verifica l'autenticità di  $cert_U$  utilizzando la sua copia di  $k_{CA}[pub]$ .
6. Decifra il crittogramma  $c$  ottenendo  $m$ :

$$m = D(c, k_V[prv])$$

7. Verifica l'autenticità della firma apposta da  $U$  su  $m$  controllando che:

$$C(f, k_U[pub]) = H(m)$$

#### Problemi

Anche questo protocollo non è esente da potenziali problemi. Infatti, il punto debole di questo meccanismo è rappresentato dai **certificati revocati** (i.e. non più validi).

Per questo motivo, le CA mettono a disposizione degli utenti un archivio pubblico contenente i certificati revocati. Si capisce che la frequenza di controllo di questi certificati e le modalità della loro comunicazione agli utenti sono cruciali per la sicurezza.

## 20.3 ECDSA

Vediamo adesso l'**Elliptic Curve Digital Signature Algorithm (ECDSA)** un algoritmo di firma digitale che fa uso delle curve ellittiche.

### 20.3.1 Funzionamento

Tale algoritmo prende come **parametri globali**:

- una **curva prima**  $E_p(a, b)$
- un **punto base**  $B$  della curva, di ordine  $n$  elevato e primo.

#### Generazione delle chiavi

Ogni utente firmatario genera una coppia di chiavi nel seguente modo:

1. Seleziona un intero positivo casuale  $d < n$ .
2. Calcola  $Q = dB$ .
3.  $k_{pub} = Q$   
 $k_{priv} = d$

#### Generazione della firma

Il firmatario Bob genera la firma nel seguente modo:

1. Seleziona un intero positivo casuale  $k < n$ .
2. Calcola il punto  $P = (x, y) = kB$ , e pone  $r = x \bmod n$ . Se  $r = 0$ , allora riparte dal punto 1.
3. Calcola  $e = H(m)$ , ove  $H$  è una funzione hash crittografica (e.g. SHA-1).
4. Calcola  $s = k^{-1}(e + dr) \bmod n$ , ove  $d$  è la sua chiave privata. Se  $s = 0$ , allora riparte dal punto 1.
5. La firma del messaggio  $m$  è la coppia  $(r, s)$ .

#### Verifica della firma

Alice riceve il messaggio  $m$  (eventualmente cifrato) e la firma  $(r, s)$ , quindi:

1. Verifica che  $r$  ed  $s$  siano interi compresi in  $[1, n - 1]$ .
2. Calcola  $e = H(m)$ .
3. Calcola  $w = s^{-1} \bmod n$ .
4. Calcola  $u_1 = ew$  e  $u_2 = rw$ .
5. Calcola il punto  $X = (x_1, y_1) = u_1B + u_2Q$ .
6. Se  $X = O$ , allora rifiuta la firma, altrimenti calcola  $v = x_1 \bmod n$ .
7. Accetta la firma se e solo se  $v = r$ .

### 20.3.2 Correttezza

Se il messaggio ricevuto da Alice è proprio quello firmato da Bob, allora:

$$s = k^{-1}(e + dr) \bmod n$$

quindi:

$$k = s^{-1}(e + dr) \bmod n = (s^{-1}e + s^{-1}dr) \bmod n = (we + wdr) \bmod n = (u_1 + u_2d) \bmod n$$

Si osserva che:

$$u_1B + u_2Q = u_1B + u_2dB = (u_1 + u_2d)B = kB$$

dunque risulta:

$$kB = u_1B + u_2Q$$

Al passo 6 della verifica, si pone:

$$v = x_1 \bmod n$$

ove  $x_1$  è l'ascissa del punto  $X = (x_1, y_1) = u_1B + u_2Q$ . Al passo 2 della generazione della firma, si pone invece:

$$r = x \bmod n$$

ove  $x$  è l'ascissa del punto  $kB$ .

La correttezza segue allora dal fatto che:

$$kB = u_1B + u_2Q$$

# Chapter 21

## Lezione 21 (29/11/2022) SSL & TLS

### 21.1 SSL

Il protocollo **SSL** (**S**ecure **S**ocket **L**ayer) fu sviluppato nel 1994 da Netscape per **effettuare comunicazioni sicure con il protocollo HTTP**. In particolare, esso è stato progettato in modo da permettere la comunicazione tra computer che **non** conoscono le reciproche funzionalità.

SSL si innesta tra un protocollo di trasporto affidabile (e.g. **TCP**) e un protocollo applicativo (e.g. **HTTP**), dal quale è completamente **indipendente**.

**Osservazione:** Il protocollo **HTTPS** è una combinazione di SSL e HTTP.

**Osservazione:** Al giorno d'oggi, SSL (con tutte le sue versioni) è considerato **obsoleto**, ed è stato rimpiazzato dal **TLS**.

#### 21.1.1 Proprietà

Caliamoci nel contesto in cui un utente  $U$  desidera accedere via Internet a un servizio offerto dal sistema  $S$ .

SSL garantisce **confidenzialità**, in quanto cifra la trasmissione mediante un **sistema ibrido**:

- cifrario asimmetrico per costruire e scambiare le chiavi di sessione.
- cifrario simmetrico che utilizza queste chiavi per criptare dati nelle comunicazioni successive.

SSL garantisce inoltre l'**autenticazione** dei messaggi, accertando l'identità dei due partner attraverso un cifrario asimmetrico o facendo uso di certificati digitali e MAC.

#### 21.1.2 Struttura

SSL è organizzato su **due livelli**:

- **SSL Handshake:** Tale livello crea un canale **sicuro, affidabile e autenticato** tra utente e sistema, entro il quale SSL Record fa viaggiare i messaggi. È quindi responsabile dell'instaurazione

e del mantenimento dei parametri usati da SSL Record.

Esso permette quindi a utente e sistema di:

- autenticarsi
- negoziare gli algoritmi di cifratura e firma
- stabilire le chiavi per i singoli algoritmi crittografici e per il MAC

- **SSL Record:** Tale livello realizza fisicamente il canale logicamente creato da SSL Handshake, rispetto al quale è quindi ad un livello di astrazione più basso. Esso è infatti connesso direttamente al protocollo di trasporto.

Obiettivo dell'SSL Record è quello di incapsulare i dati spediti dai protocolli dei livelli superiori, assicurando **confidenzialità e integrità** della comunicazione. In particolare, esso utilizza la cipher suite stabilita da SSL Handshake per cifrare e autenticare i blocchi di dati, prima di spedirli attraverso il protocollo di trasporto sottostante.

### 21.1.3 SSL Handshake

SSL Handshake consiste in uno scambio preliminare di messaggi (appunto un **handshake**) indispensabili per la creazione del canale sicuro. Attraverso tali messaggi, il sistema  $S$  (server) e l'utente  $U$  (client):

- si identificano a vicenda.
- cooperano alla costruzione delle chiavi segrete da impiegare nelle comunicazioni simmetriche successive.

Tale protocollo di handshaking prevede i seguenti passaggi:

1. **Utente - client hello:**  $U$  manda ad  $S$  un messaggio di *client hello*, con cui:
  - (a) richiede la creazione di una connessione SSL.
  - (b) specifica le prestazioni di sicurezza che desidera siano garantite durante la connessione, specificando anche cifrari e meccanismi di autenticazione che egli può supportare.

**Esempio:** Un esempio di cipher suite: **SSL\_RSA\_WITH\_AES\_CBC\_SHA1**:

- RSA per scambio chiavi di sessione
- AES per cifratura simmetrica
- CBC indica l'impiego di un cifrario a composizione di blocchi
- SHA1 funzione hash one-way per il MAC

- (c) invia una sequenza di byte casuali.

2. **Sistema - server hello:** il server  $S$ :

- (a) riceve il messaggio di *client hello*.
- (b) seleziona una cipher suite che anch'esso è in grado di supportare.
- (c) invia ad  $U$  un messaggio di *server hello* che specifica la sua scelta e contiene una nuova sequenza di byte causali.

**Osservazione:** Se  $U$  non riceve il messaggio di **server hello**, allora interrompe la comunicazione.

3. **Sistema - invio del certificato:**  $S$  si autentica con  $U$  inviandogli il proprio certificato digitale (e gli eventuali altri certificati della catena di certificazione dalla sua CA fino alla CA radice).

Se i servizi offerti da  $S$  devono essere protetti negli accessi, allora  $S$  può richiedere ad  $U$  di autenticarsi inviando il suo certificato digitale. Tuttavia, ciò avviene raramente, cosicché il  $S$  dovrà accertarsi dell'identità dell'utente in un secondo tempo.

4. **Sistema - server hello done:**  $S$  invia il messaggio *server hello done*, col quale sancisce la fine degli accordi sulla cipher suite e sui parametri crittografici a essa associati.
5. **Utente - autenticazione del sistema:** Per accettare l'autenticità del certificato ricevuto da  $S$ , l'utente  $U$  controlla che:

- la data corrente sia inclusa nel periodo di validità del certificato.
- la CA che ha firmato il certificato sia tra quelle “fidate”, cioè tra quelle di cui conosce la chiave pubblica.
- la firma apposta dalla CA sia autentica.

6. **Utente - invio del pre-master secret e costruzione del master secret:** L'utente  $U$ :
  - (a) Costruisce un **pre-master secret** costituito da una nuova sequenza di byte casuali (i.e. genera un numero segreto).
  - (b) Lo cifra con il cifrario a chiave pubblica su cui si è accordato con  $S$ .
  - (c) Spedisce il relativo crittogramma a  $S$ .
  - (d) Il pre-master secret viene combinato da  $U$  con alcune stringhe note e con i byte casuali presenti sia nel messaggio di *client hello* che in quello di *server hello*.
  - (e) A tutte queste sequenze,  $U$  applica delle funzioni hash one-way (SHA-1 e MD5) secondo una combinazione opportuna, ottenendo così il **master secret**.

7. **Sistema - ricezione del pre-master secret e costruzione del master secret:** Il sistema  $S$ :

- (a) Decifra il crittogramma contenente il pre-master secret inviato da  $U$ .
- (b) Calcola il master secret mediante le stesse operazioni eseguite da  $U$  al passo 6, dato che dispone delle stessa informazioni.
8. **Utente - invio del certificato (opzionale):** Se all'utente  $U$  viene richiesto un certificato (al passo 3) ed egli non lo possiede, allora il sistema interrompe l'esecuzione del protocollo. Altrimenti,  $U$  invia il proprio certificato con allegate una serie di informazioni firmate con la sua chiave privata, tra cui:
  - (a) master secret
  - (b) messaggi scambiati fino a quel momento (**SSL-history**).

Prevedibilmente,  $S$  controlla il certificato di  $U$  e verifica l'autenticità e correttezza della SSL-history. In presenza di anomalie, la comunicazione con  $U$  viene interrotta.

9. **Utente/Sistema - messaggio finished:** Il messaggio *finished* è il primo messaggio protetto mediante il master secret e la cipher suite su cui i due partner si sono accordati. Esso viene prima costruito da  $U$  e spedito ad  $S$ , poi costruito da  $S$  e spedito ad  $U$ .

Nei due invii, la struttura del messaggio è la stessa, ma cambiano le informazioni in esso contenute. La costruzione avviene in due passi:

- Si concatenano il master secret, tutti i messaggi di handshake scambiati fino a quel momento e l'identità del mittente.
- La stringa ottenuta viene trasformata applicando le funzioni SHA-1 e MD5. Si ottiene così una coppia di valori che costituisce il messaggio *finished*.

**Osservazione:** Il messaggio è diverso nelle due comunicazioni perché  $S$  aggiunge ai messaggi di handshake anche il messaggio *finished* ricevuto da  $U$ .

**Osservazione:** Il destinatario della coppia non può invertire la computazione precedente, in quanto generata da funzioni one-way. Esso deve quindi ricostruire l'ingresso delle due funzioni SHA-1 e MD5 (cosa comunque immediata), le ricalcola e controlla che la coppia generata coincida con quella ricevuta.

Il master secret è utilizzato da  $U$  ed  $S$  per costruire una propria **tripla** contenente:

- la **chiave segreta** da adottare nel cifrario simmetrico
- la **chiave di autenticazione** del messaggio (costruzione del MAC)
- la **sequenza di inizializzazione** per cifrare in modo aperiodico messaggi molto lunghi (e.g. usata come valore iniziale nel CBC).

**Osservazione:** Le triple di  $U$  ed  $S$  sono diverse tra loro, ma comunque note a entrambi i partner. Ciascuno usa la propria tripla al fine di aumentare la sicurezza della comunicazione.

#### 21.1.4 SSL Record

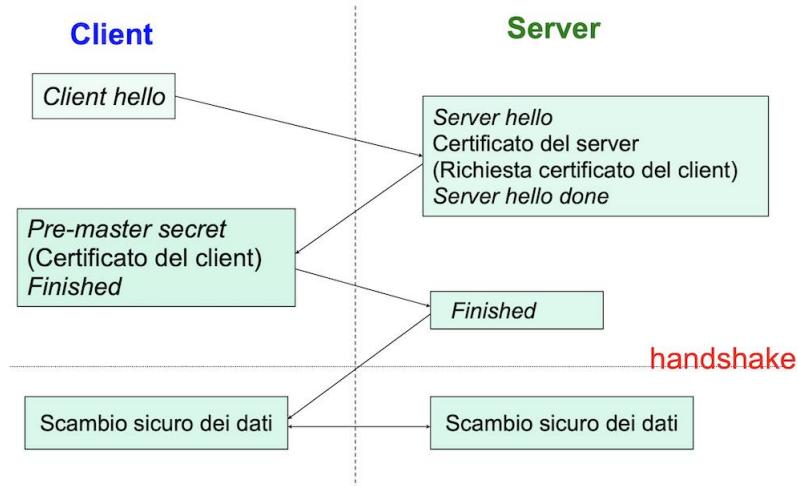
Il canale sicuro approntato dal protocollo SSL Handshake viene realizzato dal protocollo SSL Record.

Tale protocollo prevede che i dati siano frammentati in **blocchi**, e che ciascun blocco venga:

1. **numerato** → **compresso** → **autenticato** mediante l'aggiunta di un MAC.
2. **cifrato** mediante il cifrario simmetrico su cui  $U$  ed  $S$  si sono accordati.
3. **trasmesso** utilizzando il protocollo di trasporto sottostante.

Prevedibilmente, il destinatario esegue un procedimento inverso sui blocchi ricevuti:

1. **decifra e verifica** la loro integrità attraverso il MAC.
2. **decomprime e riassembla** i blocchi in chiaro.
3. li **consegna** all'applicazione sovrastante.



### 21.1.5 Sicurezza

Nei passi di hello i due partner creano e si inviano due sequenze casuali per la costruzione del master secret, che risulta così diverso in ogni sessione di SSL. Questo implica che un crittoanalista attivo non può riutilizzare i messaggi di handshake catturati sul canale per sostituirsi a  $S$  in una successiva comunicazione con  $U$ .

Inoltre, SSL Record numera in modo incrementale ogni blocco di dati, che autentica attraverso un MAC. Per prevenire la modifica del blocco da parte di un crittoanalista attivo, il MAC viene calcolato come **immagine hash one-way** di una stringa costruita concatenando:

- contenuto del blocco
- numero del blocco nella sequenza
- chiave del MAC
- alcune stringhe note e fissate a priori

Dato che i MAC sono cifrati insieme al messaggio, un crittoanalista non può alterarli senza avere forzato prima la chiave simmetrica di cifratura, quindi un attacco volto a modificare la comunicazione tra i due partner è difficile almeno quanto quello volto alla sua decriptazione!

Ancora, il canale definito da SSL Handshake è immune da attacchi man-in-the-middle, poiché il sistema  $S$  viene autenticato con un certificato digitale. Infatti, l'utente  $U$  può comunicare il pre-master secret al sistema  $S$  in modo sicuro attraverso la chiave pubblica presente nel certificato di  $S$ . Solo  $S$  può quindi decifrare quel crittogramma e costruire il master secret, su cui si fonda la costruzione di tutte le chiavi segrete adottate nelle comunicazioni successive. Di conseguenza, solo  $S$  potrà entrare nella comunicazione con  $U$ .

**Osservazione:** L'autenticazione opzionale di  $U$  non rappresenta una debolezza, dato che comunque l'utente potrà essere autenticato dal sistema in seguito. Utente e password suggeriscono nulla?

Infine, le tre sequenze casuali generate da  $U$  e da  $S$  e comunicate nei messaggi di *client hello*, *server hello*, pre-master secret sono usate per creare il master secret, quindi per generare le chiavi segrete

di sessione. La sequenza corrispondente al pre-master secret viene generata da  $U$  e comunicata per via **cifrata** ad  $S$ .

**Osservazione:** La **non predicitività** di questa sequenza è cruciale per la sicurezza del canale SSL. Infatti, una sua cattiva generazione renderebbe il protocollo molto debole.

## 21.2 TLS

Il protocollo **TLS (Transport Layer Security)** deriva direttamente da SSL, essendone una variante più sicura.

La caratteristica chiave di TLS, è che permette a client e server di stabilire un insieme di chiavi simmetriche condivise, da utilizzare per cifrare e autenticare la sessione di comunicazione.

**Nota:** Daremo di solo una descrizione semplificata e ad alto livello di **TLS 1.3**.

### 21.2.1 Proprietà

TLS permette al client e al server di **autenticarsi reciprocamente**, anche se è spesso utilizzato solo per autenticare il server. Infatti, l'autenticazione del client, se necessaria, avviene quasi sempre quando la sessione TLS è avviata, ad esempio tramite utente/password.

### 21.2.2 Struttura

Come SSL, TLS consiste di due parti:

- **Protocollo di Handshake:** protocollo di scambio di chiavi per stabilire le chiavi simmetriche condivise.
- **Protocollo Record-layer:** utilizza le chiavi condivise per cifrare e autenticare la comunicazione.

### 21.2.3 Protocollo di Handshake

Il client  $U$  possiede un **insieme di chiavi pubbliche** di CA ( $pk_1, pk_2, \dots$ ).

Il server  $S$  possiede invece una coppia chiave pubblica/chiaia privata ( $pk_S, sk_S$ ) per la firma digitale. Possiede inoltre un certificato digitale  $cert_S$  per  $pk_S$ , rilasciato da una delle CA di cui  $U$  ha la chiave pubblica.

L'handshaking prevede quindi i seguenti passi:

1.  $U$  invia ad  $S$  il messaggio iniziale del protocollo DH per lo scambio delle chiavi, che include:
  - la specifica del gruppo  $G$  usato dal client, il quale può essere  $\mathbb{Z}_p^*$ , oppure una curva ellittica.
  - il generatore  $g$ , oppure il punto  $B$  della curva ed il relativo ordine.
  - il valore  $g^x$ , oppure il punto  $xB$ , calcolato usando un intero segreto  $x$  scelto casualmente da  $U$ .
  - un **nonce**  $N_U$  (sequenza casuale di bit).

- informazioni sulle cipher suites che  $U$  è in grado di supportare.
2. Il server  $S$ :
- (a) Completa il protocollo DH inviando un messaggio al client che contiene  $g^y$ , oppure il punto  $yB$ , calcolato usando un intero segreto  $y$  scelto casualmente da  $S$ .
  - (b) Invia un suo nonce  $N_S$ .
  - (c) Calcola  $K = g^{xy}$ , oppure  $K = xyB$ , e applica una **key derivation function** per estrarre da  $K$  le chiavi  $K_S', K_U', K_S, K_U$  per una **cifratura autenticata**.
  - (d) Invia ad  $U$  i seguenti dati, cifrati con  $K_S'$ :
    - la propria **chiave pubblica**  $pk_S$
    - il **certificato**  $cert_S$
    - la **firma**  $\sigma$  calcolata usando la chiave privata  $sk_S$  su tutti i messaggi di handshake inviati
3. Il client  $U$ :
- (a) Calcola  $K = g^{xy}$ , oppure  $K = xyB$ , e deriva le chiavi  $K_S', K_U', K_S, K_U$ .
  - (b) Usa  $K_S'$  per recuperare  $pk_S, cert_S$  e  $\sigma$ .
  - (c) Se possiede la chiave pubblica della CA che ha rilasciato  $cert_S$ , allora verifica il certificato. Altrimenti interrompe la comunicazione.
  - (d) Verifica la firma  $\sigma$  sui messaggi di handshake scambiati usando  $pk_S$ .
  - (e) Calcola il MAC dei messaggi di handshake scambiati usando  $K_U'$ , quindi lo invia ad  $S$ .

#### 21.2.4 Protocollo Record-layer

Se l'handshaking va a buon fine, allora si procede al protocollo Record-layer:

- $U$  usa  $K_U$  per cifrare i messaggi che invia ad  $S$ .
- $S$  usa  $K_S$  per cifrare i messaggi che invia ad  $U$ .

#### 21.2.5 Sicurezza

Alla fine del protocollo di Handshake,  $U$  ed  $S$  condividono le chiavi di sessione  $K_U$  e  $K_S$ , che possono usare per cifrare e autenticare la comunicazione.

**Osservazione:** Si noti come  $K_U'$  e  $K_S'$  sono usate solo per la fase di Handshake.

Dato che  $U$  verifica il certificato, sa che  $pk_S$  è la chiave pubblica corretta di  $S$ . Inoltre, se la firma  $\sigma$  è valida, allora  $U$  conclude che sta comunicando con  $S$ . Allo stesso modo,  $S$  ha firmato tutti i messaggi scambiati per l'esecuzione del protocollo DH, cosicché  $U$  è certo che nessun valore è stato modificato.

Infine, alla fine della fase di Handshake,  $U$  sa che condivide le chiavi  $K_C$  e  $K_S$  con il legittimo  $S$ .

### 21.2.6 TLS 1.3 vs TLS 1.2

Noi abbiamo visto il funzionamento di TLS 1.3. In TLS 1.2 client e server potevano stabilire la chiave  $K$  usando un cifrario a chiave pubblica al posto del protocollo DH. In realtà,  $K$  era stabilita solo dal client, che la comunicava poi al server.

Questo comportamento non è più possibile in TLS 1.3, al fine di garantire la **forward secrecy**, cioè la segretezza delle chiavi di sessione precedenti nel caso di un server compromesso. In particolare, La forward secrecy è una proprietà dei protocolli di negoziazione delle chiavi che assicura che se una chiave di cifratura a lungo termine viene compromessa, le chiavi di sessione generate a partire da essa rimangono riservate.

In tale contesto, DH garantisce forward secrecy, dato che il valore  $y$  del server, usato nel protocollo di Handshake, può essere cancellato alla fine del protocollo. Infatti, senza  $y$  il crittoanalista non può ricostruire la chiave di sessione  $K$ .

Di contro, usando un cifrario a chiave pubblica, **non** si ha forward secrecy, dato che la chiave privata del server non può essere cancellata. Perciò, se un avversario la ottiene, può decifrare i critogrammi scambiati nelle esecuzioni passate del protocollo di Handshake e recuperare le chiavi di sessione usate dalle parti coinvolte.

## 21.3 Zero knowledge

In generale, i protocolli zero-knowledge prevedono la presenza di un **prover (P)** e di un **verifier (V)**. Il prover afferma di sapere qualcosa, e lo deve dimostrare al verifier senza rivelargli la conoscenza di quella cosa.

### 21.3.1 Principi

I protocolli zero-knowledge rispettano i seguenti principi:

- **Completezza:** se P è onesto, cioè la sua affermazione è vera, allora V accetta sempre la dimostrazione.
- **Correttezza:** se P è disonesto, cioè la sua affermazione è falsa, allora V può essere ingannato con probabilità minore di  $\frac{1}{2^k}$ , over  $k$  è stabilito da V.
- **Zero-knowledge:** se P è onesto, V (anche se disonesto) non può acquisire alcune informazioni su P oltre alla veridicità della sua affermazione.

Si capisce quindi che i protocolli zero-knowledge sono di fondamentale importanza quando chi si deve identificare a qualcuno non si fida di quel qualcuno, e quindi non vuole dargli alcuna informazione all'infuori della propria identità.

## 21.4 Protocollo di Fiat-Shamir

Il protocollo zero-knowledge di identificazione di Fiat-Shamir basa la sua sicurezza sul problema del **calcolo della radice in modulo**.

### 21.4.1 Setup

Vi è anzitutto una fase preliminare in cui P, cioè chi si deve identificare, crea una coppia di chiavi pubblica e privata, nel seguente modo:

1. Sceglie due numeri primi grandi  $p, q$  e calcola  $n = pq$ ;
2. Sceglie casualmente un intero  $s < n$ ;
3. Calcola quindi  $t = s^2 \pmod{n}$ .

P produce in questo modo la coppia di chiavi:

- $k_{pub} = (t, n)$
- $k_{priv} = (p, q, s)$

Osserviamo quindi che ogni utente possiede la sua personale coppia di chiavi.

### 21.4.2 Funzionamento

Il funzionamento del protocollo si basa sul fatto che V chiede ciclicamente a P di risolvere dei problemi matematici che possono essere risolti solamente se P conosce la chiave segreta di chi dichiara di essere.

Quindi, se P è onesto, allora usa la sua chiave priata ed è in grado di risolvere ogni problema sottopostogli da V. Se P è disonesto, cioè finge di essere qualcun'altro, allora non è in grado di risolvere i problemi sottopostigli da V, dato che servirebbe la chiave privata di chi dichiara di essere al fine di risolverli.

In particolare, il protocollo funziona nel seguente modo:

Per il numero  $k$  di volte che vuole V, si ripete ciclicamente i seguenti passi:

1. V chiede a P di iniziare una nuova iterazione;
2. P genera casualmente un intero  $r < n$ , quindi invia  $u = r^2 \pmod{n}$  a V;
3. V genera casualmente un bit  $e$  che invia quindi a P;
4. P calcola a questo punto  $z = rs^e \pmod{n}$  e lo invia a V;
5. V calcola quindi  $x = z^2 \pmod{n}$ ;
6. Se  $x = ut^e \pmod{n}$ , allora V può procedere con l'iterazione successiva (del tutto analoga) o concludere l'identificazione di P con successo, altrimenti l'identificazione fallisce.

### 21.4.3 Completezza

È immediato dimostrare la completezza del protocollo, dato che se P è onesto, cioè conosce  $s$ , allora sarà ad ogni iterazione in grado di risolvere correttamente il problema postogli da V. Quindi V identifica sempre un utente onesto.

#### 21.4.4 Correttezza

Si noti che se P finge di essere un altro utente Q, allora conoscerà solamente la chiave pubblica di Q, ma **non** la sua chiave privata. Di conseguenza, P non sarà in grado di risolvere i problemi postigli da V.

Ad ogni iterazione, P può provare a raggiungere la soluzione del problema cercando al passo **2** di indovinare quale sarà il valore del bit  $e$  che gli invierà V. Infatti:

- **Se pensa di ricevere 0**, allora calcola  $u = r^2 \pmod n$ .
- **Se pensa di ricevere 1**, allora calcola  $u = r^2 t^{-1} \pmod n$ .

In ogni caso, calcola  $z = r$  (per forza, dato che non conosce  $s$ ).

Si osservi a questo punto che, siccome  $e$  viene generato casualmente, P ha probabilità  $\frac{1}{2}$  di indovinare. Dato che V itera  $k$  volte, la probabilità che P indovini tutte le volte è  $\frac{1}{2^k}$ . Infatti, basta che P non indovini una sola volta per far sì che V si accorga che esso è un impostore.

#### 21.4.5 Zero knowledge

Anche se non ne vediamo la dimostrazione (in quanto molto complessa), questo protocollo fa sì che V non possa in alcun modo acquisire informazioni sulla chiave privata di P, e quindi la può ricavare solamente risolvendo il problema del calcolo della radice in modulo. Questo è ciò che rende per l'appunto il protocollo zero knowledge.

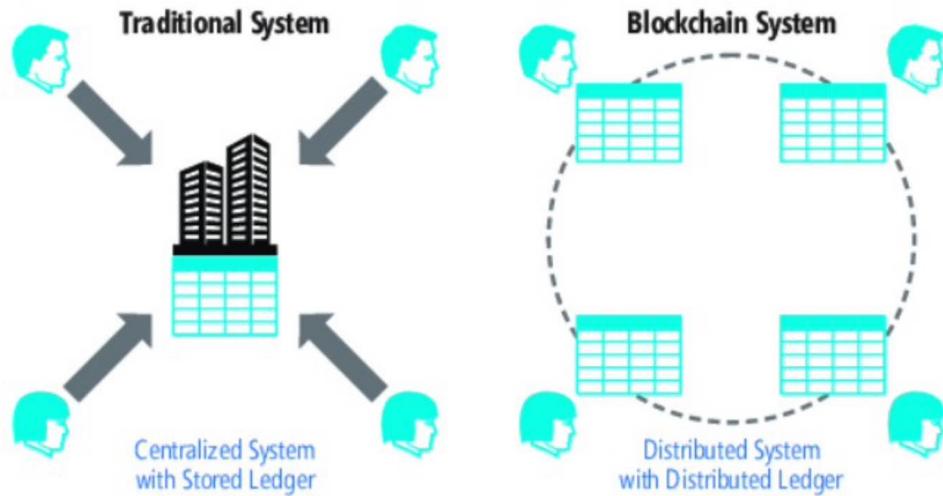
## Chapter 22

# Lezione 22 (01/12/2022) Blockchain

Il concetto di **Blockchain** è nato insieme a **Bitcoin**, con lo scopo di creare un sistema di scambio di denaro completamente decentralizzato. Nelle blockchain, vi è perciò l'assenza di terze parti fidate.

La blockchain si basa sul concetto di **ledger**, cioè di registri che memorizzano tutte le transazioni che avvengono all'interno del sistema. Se in un sistema centralizzato vi è un ente centralizzato che si occupa di mantenere un unico ledger, nella blockchain **ogni** utente mantiene una sua copia del ledger, idealmente uguale a quelle di tutti gli altri utenti.

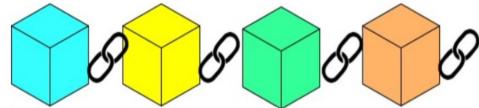
**Osservazione:** Si capisce che proprietà fondamentale di questo **distributed ledger** è la **tamper freeness**, cioè non deve poter essere manomesso!



Si capisce che la blockchain è un sistema **peer-to-peer**, in quanto tutti hanno stesse informazioni e stessi diritti.

## 22.1 Struttura

Il distributed ledger è di fatto la blockchain in sé per sé. Si parla di **block-chain** in quanto il ledger è formato da una catena di blocchi.



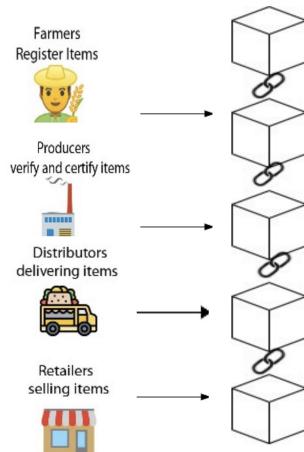
**Osservazione:** Siccome l'intera blockchain è ormai svariate centinaia di GB, l'utente medio della blockchain memorizza solo i blocchi più recenti.

Ogni blocco contiene:

- **Dati** relativi a transazioni di denaro, contratti, certificati di proprietà, etc.
- **Valore hash**
- **Valore hash del blocco precedente**

### 22.1.1 Dati

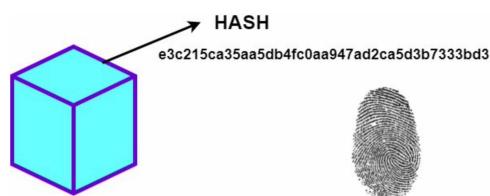
Come abbiamo accennato, in un blocco della blockchain si può registrare virtualmente qualunque tipo di informazione.



In generale, le informazioni aggiunte con maggior frequenza sono le **transazioni** di criptovalute o asset finanziari.

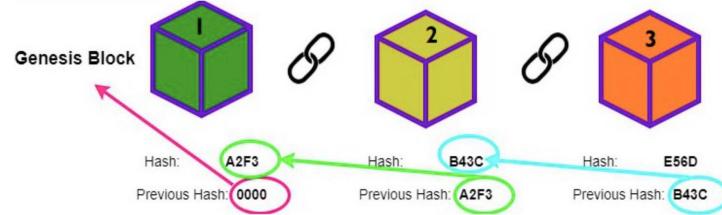
### 22.1.2 Hash

Ogni blocco mantiene un proprio valore hash che fa da **fingerprinting**.

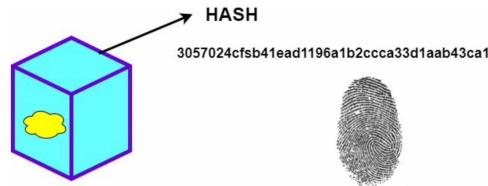


Il valore hash del blocco viene calcolato sull'intero contenuto e sull'**hash del blocco precedente**.

**Osservazione:** Il primo blocco creato, ha preso come hash del blocco precedente un valore deciso a tavolino.



Questo significa che se anche una minima parte del blocco viene alterata, l'hash cambia completamente il suo valore.



### 22.1.3 Proprietà

La blockchain deve soddisfare le seguenti proprietà essenziali:

- **Distribuzione del controllo:** chiunque può aggiungere blocchi alla blockchain.
- **Tamper freeness:** nessuno può modificare una registrazione scritta in precedenza, e neppure aggiungere una registrazione tra due già esistenti.
- **Trasparenza:** tutti possono verificare la correttezza delle transazioni e leggerne il contenuto.
- **Consistenza:** un blocco viene aggiunto alla blockchain solo quando tutti i nodi hanno acconsentito al suo inserimento nella blockchain.

## 22.2 Sicurezza

Per ogni blocco, mantenere il proprio hash e quello del blocco precedente garantisce che la blockchain non venga alterata in un secondo momento (**tamper freeness**).

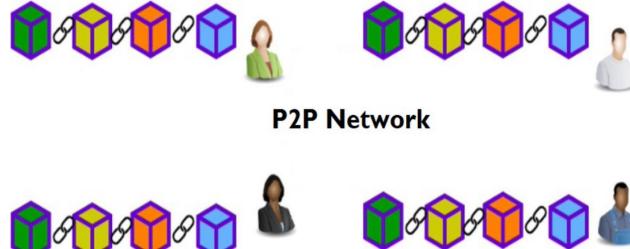
Infatti, cambiare un hash causa la modifica di tutti i blocchi successivi.



Ovviamente, per rendere il sistema sicuro deve essere **computazionalmente complesso ricalcare l'hash di un blocco**. Questa garanzia viene offerta in Bitcoin mediante il meccanismo di **Proof of Work** (more on this later).

## 22.3 Aggiunta di un blocco

Come abbiamo detto, ogni utente ha la sua copia (eventualmente parziale) della blockchain.



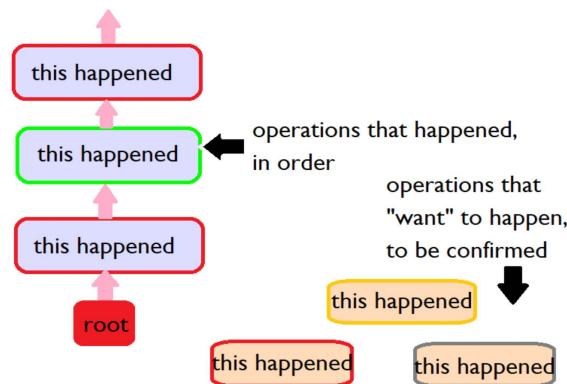
Questo porta a farci una domanda: chi decide quale blocco aggiungere alla blockchain?

Come abbiamo detto, un blocco viene aggiunto alla blockchain solo quando tutti i nodi hanno acconsentito al suo inserimento nella blockchain. Quindi vi deve essere un **consenso distribuito**. Tuttavia, bisogna fare in modo che nodi maliziosi non possano bloccare l'intero funzionamento del meccanismo. Per questo motivo, l'algoritmo di consenso deve operare in modo più complesso.

**Nota:** Per semplicità, supporremo da adesso che ogni blocco contenga una sola transazione.

### 22.3.1 Meccanismo di consenso

In generale, abbiamo ad un dato istante un insieme di transazioni, delle quali solo quelle confermate dall'algoritmo di consenso appartengono alla blockchain.



In tale contesto, il **consenso** è il meccanismo tramite il quale si stabilisce:

- chi decide quale transazione verrà aggiunta alla blockchain;
- quale transazione, tra quelle validate, verrà aggiunta alla blockchain.

## Maggioranza

Essenzialmente, il sistema blockchain funziona correttamente finché la *maggioranza* è onesta, intesa come la maggioranza del **potere computazionale**, e **non** la maggioranza degli utenti.



In Bitcoin, tutti partecipano ad una sorte di *lotteria (mining)*. In tale contesto, solo chi vince (e vincere è complesso), può appendere il prossimo blocco alla blockchain. In generale, vincere la lotteria è computazionalmente molto difficile. Di conseguenza, chi ha maggiore potere computazionale ha più probabilità di vincere.

## Consenso di Nakamoto

Il consenso di Nakamoto funziona finché il  $50\% + 1$  vince un nodo onesto. Esso opera nel seguente modo:

1. ad ogni round: seleziona un nodo in modo casuale, nella maggior parte dei casi il nodo è onesto;
- Osservazione:** In Bitcoin la probabilità che un nodo sia selezionato è proporzionale alla sua capacità computazionale e la selezione viene fatta mediante Proof of Work. I nodi che tentano di *risolvere* la Proof of Work sono detti *miners*, mentre il processo di validazione viene riferito come *mining*.
2. il nodo selezionato propone, in modo unilaterale, senza contattare altri nodi, il prossimo blocco di transazioni da inserire nella blockchain;
3. il blocco viene mandato in broadcast a tutti i nodi della rete;
4. tutti i nodi controllano la validità del blocco ed aggiornano la loro copia della blockchain.

## 22.4 Proof of Work

La Proof of Work è un meccanismo che consente ad una entità di provare ad un'altra entità che si è impiegata una certa quantità di risorse computazionali per un certo periodo.

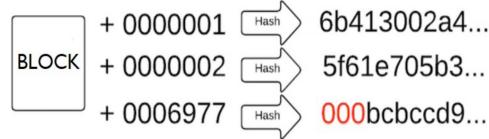
Essa è basata su puzzle crittografici che richiedono un tempo considerevole per essere risolti. Le caratteristiche sono:

- Non esistono scorciatoie per risolverli prima, quindi richiedono tempo esponenziale anche con la computazione quantistica;
- Difficile è trovare la soluzione al puzzle, ma verificarla è facile per chiunque.

Nel nostro caso, chi risolve il puzzle può decidere il prossimo blocco da inserire nella blockchain.

### 22.4.1 Funzionamento

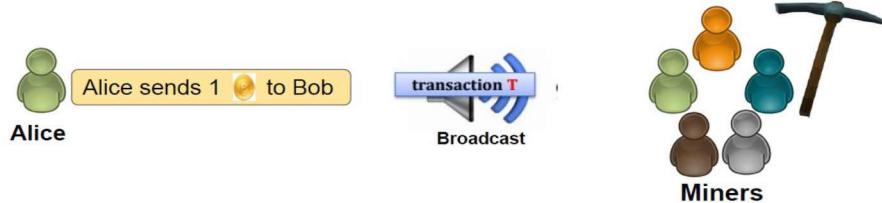
In Bitcoin, la proof of work consiste nel trovare un valore  $x$  tale che l'hash del blocco insieme ad  $x$  abbia le prime  $k$  cifre a 0.



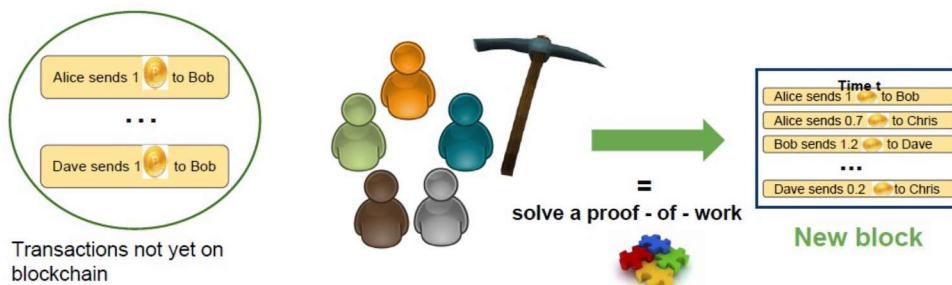
**Osservazione:** È di immediata comprensione che il problema diventa sempre più difficile all'aumentare di  $k$ .

### 22.4.2 Recap

Ricapitolando, questo è uno schema di quanto accade per aggiungere una transazione sulla blockchain:



I miners competono in una gara per decidere chi inserirà il prossimo blocco



I found a new block!



**Osservazione:** Come si può vedere, ogni blocco contiene in realtà molte transazioni.

## 22.5 Smart contracts

I contratti tradizionali vengono stipulati tra due parti mediante l'ausilio di una terza parte fidata che garantisce i termini del contratto.

L'introduzione degli **smart contract** rimuove la necessità di una terza parte e automatizza l'esecuzione del contratto. In questo modo si ottiene la **disintermediazione**, cioè non è richiesto alcun intermediario.

### 22.5.1 Realizzazione

Lo smart contract è un codice scritto sulla blockchain. Quindi un nodo effettua il deploy, e tutte le parti interessate possono accedere al contratto.

Tale contratto è eseguito da tutti i nodi della blockchain.

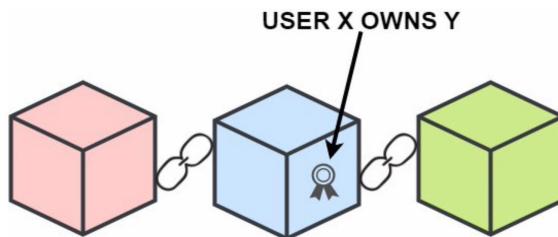
#### Esempio

Bob è all'aeroporto ed il suo volo è in ritardo, però possiede una assicurazione che gli garantisce un rimborso nel caso di ritardi del volo. La compagnia di assicurazione ha generato uno smart contract su una blockchain (e.g. Ethereum) che è connesso al database delle compagnie aeree e monitora i ritardi dei voli. Quindi, non appena viene verificata una condizione di ritardo di almeno X minuti, esso genera automaticamente un rimborso in criptomoneta che memorizza nel wallet di Bob.

## 22.6 NFT

Un **token non fungibile** (NFT) è un oggetto che non può essere replicato né copiato, in quanto unico nel suo genere.

Gli NFT vivono sulla blockchain all'interno di smart contracts, e rappresentano il possesso di un oggetto unico.



Di conseguenza, chi acquista un'opera digitale NFT, di fatto sta acquistando i **digital bragging rights**, cioè un certificato di proprietà dell'opera. Del resto, gli NFT sono risorse online che chiunque può scaricare ed avere. Tuttavia, solo l'owner può possedere il certificato di proprietà dell'opera.

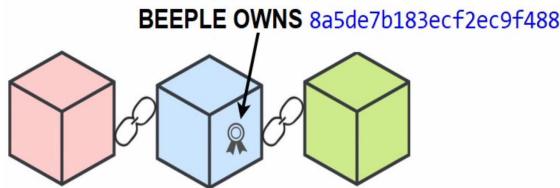
### 22.6.1 Struttura

Un NFT contiene:

- **Identificatore univoco;**
- **Nome del token;**
- **Simbolo del token.**

**Esempio**

- ID: 8a5de7b183ecf2ec9f488
- Nome: everydays: first 5000 days
- Simbolo: EF5000D



**Osservazione:** L'NFT non contiene l'opera stessa, ma un link all'opera. Questo è anche dovuto per ragioni di spazio occupato.

# Chapter 23

## Lezione 23 (05/12/2022) Crittografia quantistica

Naturalmente, la crittografia quantistica basa la sua sicurezza su proprietà della fisica quantistica.

In particolare, basiamo il nostro discorso successivo sulla conoscenza di 4 proprietà fondamentali:

- **Sovrapposizione:** proprietà di un sistema quantistico di trovarsi in più stati contemporaneamente.
- **Decoerenza:** la misurazione di un sistema quantistico gli fa perdere la sovrapposizione degli stati con conseguente collasso ad uno stato singolo.
- **No-cloning:** impossibilità di duplicare un sistema quantistico non noto, conservando nella copia lo stato del sistema originale.
- **Entanglement:** possibilità che due o più elementi si trovino in stati quantici correlati tra di loro, in modo che, pur se posti a grandi distanze, mantengono la correlazione.

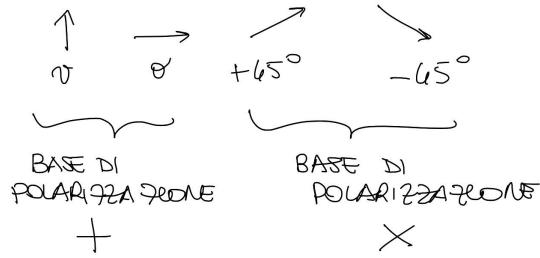
Detto ciò, passiamo a vedere come funziona uno degli algoritmi crittografici di scambio delle chiavi più noti, cioè il BB84.

### 23.1 Protocollo BB84

Il protocollo BB84 prevede uno scambio delle chiavi mediante invio di **foton polarizzati**.

In particolare, un fotone può trovarsi in uno di 4 possibili stati di polarizzazione, definiti come il piano di oscillazione del suo campo elettrico. Tali stati sono divisi in due **basi di polarizzazione**:

- **Base di polarizzazione ortogonale (+):**
  - $\theta = 90^\circ (\uparrow)$  - **0** logico.
  - $\theta = 0^\circ (\rightarrow)$  - **1** logico.
- **Base di polarizzazione diagonale ( $\times$ ):**
  - $\theta = 45^\circ (\nearrow)$  - **0** logico.
  - $\theta = -45^\circ (\searrow)$  - **1** logico.



Quindi, riassumendo abbiamo:

- **0 logico:**  $\uparrow, \nearrow$
- **1 logico:**  $\rightarrow, \searrow$

**Osservazione:** Non è possibile capire lo stato del fotone se la base di polarizzazione non è nota. Quindi, l'unica misura possibile è tra 2 stati prtoponali della stessa base.

### 23.1.1 Apparecchiature utilizzate

#### OPG

Il **one-photon gun** (OPG) consente di generare ed emettere un fotone alla volta, con una **polarizzazione prestabilita**.

#### PC

La **cella di Pockels** (PC) consente di imporre la polarizzazione di un fotone agendo su un campo elettrico di controllo.

#### PBS

Il **beam splitter polarizzante** (PBS) dirotta i fotoni in ingresso verso una tra due uscite A od R per un fenomeno ottico rispettivamente di attraversamento o riflessione.

IL PBS ha un **asse di polarizzazione  $\mathcal{S}$** . Un fotone in ingresso polarizzato in direzione  $\mathcal{F}$  viene inviato all'uscita A con probabilità  $\cos^2 \theta$  (e quindi ad R con probabilità  $\sin^2 \theta$ ) ove  $\theta$  è l'angolo compreso tra  $\mathcal{F}$  ed  $\mathcal{S}$ .

Nel primo caso il fotone esce da A con la polarizzazione  $\mathcal{S}$  del PBS, mentre nel secondo caso esce da R con polarizzazione ortogonale ad  $\mathcal{S}$ . Si distinguono dunque i seguenti casi:

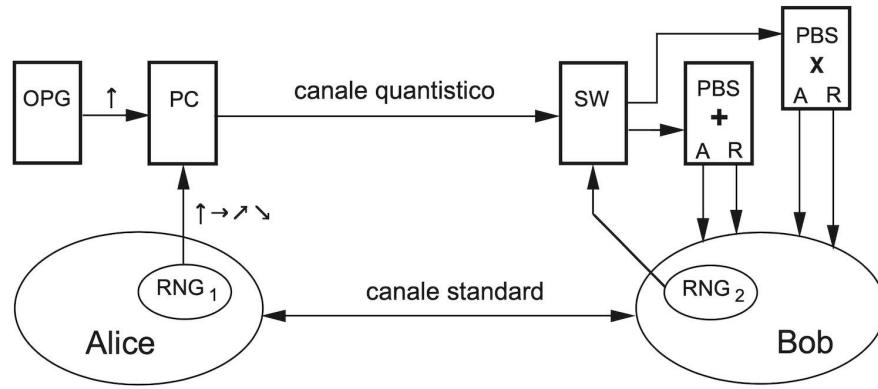
- $\mathcal{F} = \mathcal{S} \implies \theta = 0 \implies \cos^2 \theta = 1 \implies$  il fotone esce sicuramente da A.
- $\mathcal{F} \perp \mathcal{S} \implies \theta = 90 \implies \sin^2 \theta = 1 \implies$  il fotone esce sicuramente da R.
- $\theta = \pm 45 \implies \sin^2 \theta = \cos^2 \theta \implies$  il fotone può uscire da A o da R con pari probabilità.

bit e fotone inviato da A			
basi di B	0 ↑	0 ↗	1 →
+	↑	↑ →	→
×	↗ ↓	↗	↗ ↓

**Osservazione:** Una volta che il fotone esce dal PBS, non si può in alcun modo risalire al suo precedente stato quantico.

### 23.1.2 Scambio generale delle chiavi

Una delle possibili strutture circuituali per eseguire il protocollo tra i partner Alice e Bob è la seguente:



Essa prevede il seguente funzionamento:

1. Alice trasmette a Bob una sequenza  $S$  di bit sotto forma di fotoni, stabilendo **a caso** la base di polarizzazione per ciascuno di essi.

**Osservazione:** Alice stabilisce la polarizzazione dei fotoni controllando la cella PC posta all'ingresso della linea di trasmissione.

2. Bob stabilisce a caso una base di polarizzazione controllando un deviatore che dirotta il fotone in arrivo su uno di due moduli PBS orientati  $+$  e  $\times$ , e legge il bit ricevuto a seconda che il fotone provenga dall'uscita A od R del PBS interessato.

**Osservazione:** Le basi impiegate da Alice e Bob sono scorrelate e coincidono in media nella metà dei casi, stima significativa poiché  $S$  è molto lunga. Dunque, solo metà dei bit di  $S$  mantengono la polarizzazione scelta da Alice dopo l'intervento di Bob e sono letti correttamente da esso, mentre le letture relative alle basi diverse di Alice e Bob non sono significative.

3. Alice e Bob devono ora decidere quali sono i bit corretti. Per eseguire la scelta dei bit Alice e Bob devono scambiarsi alcune informazioni addizionali per cui possono utilizzare un canale di trasmissione standard.

**Osservazione:** La comunicazione sul canale standard può essere in chiaro, ma deve essere autenticata.

### 23.1.3 Funzionamento di BB84

Avendo visto in generale come avviene lo scambio, vediamo ora nello specifico come opera BB84:

1. Alice ( $A$ ) invia la sequenza codificata  $S_A$  sul canale quantistico.
2. Bob ( $B$ ) interpreta  $S_A$  con le basi che ha scelto, ottenendo una sequenza  $S_B$  che coincide con  $S_A$  per circa metà degli elementi.
3. Usando il canale standard, Bob comunica ad Alice le basi che ha scelto e Alice gli indica quali basi sono comuni alle sue.
4. In assenza di interferenze sul canale quantistico da parte di un crittoanalista attivo Eve, Alice e Bob possiedono una sottosequenza identica  $S'_A = S'_B$  relativa ai bit codificati e decodificati con le basi comuni che potrebbe costituire la chiave.

#### Esempio

1	2	3	4	5	6	7	...	successione temporale
1	0	1	1	1	0	0	...	$S_A$
+	x	+	x	x	+	x	...	basi di $A$
→	/	→	\	\	↑	/	...	invio di $A$
+	x	+	+	+	x	x	...	basi di $B$
→	/	→	↑	→	\	/	...	lettura di $B$
1	0	1	0	1	1	0	...	$S_B$

#### Crittoanalista Eve

Il crittoanalista Eve potrebbe intercettare la sequenza spedita da Alice, interpretarla e rispedirla a Bob. Per fare questo dovrebbe usare una sequenza di basi per la lettura inevitabilmente scorrelata dalla sequenza di basi scelta da Alice che Eve non conosce, e questo distruggerebbe parte dell'informazione contenuta nella sequenza  $S_A$ . In conseguenza Bob, interpretando la sequenza di fotoni ricevuta da Eve, otterrà una sequenza  $S_B$  che coincide con la  $S_A$  in meno elementi di quelli di  $S'_B$ , perché questi saranno in parte corrotti quando vengono rielaborati da Eve con basi diverse. Dunque il protocollo prosegue effettuando il passo:

5. Alice e Bob sacrificano una parte  $S''_A, S''_B$  delle sequenze  $S'_A, S'_B$  in posizioni prestabilite, comunicandole sul canale standard. In caso di errore, cioè se  $S''_A \neq S''_B$ , i due partner interrompono lo scambio di chiave perché vi è evidenza di intercettazione o comunque di malfunzionamento. In caso di assenza di errore,  $S'_A - S''_A = S'_B - S''_B$  è adottata come chiave.

#### Esempio

La sottosequenza  $S'_B$  calcolata da Bob con le basi comuni ad Alice, indicata in grassetto, differisce dalla  $S'_A$ , infatti  $S'_B(2) \neq S'_A(2)$ . Posto che Alice e Bob concordino di sacrificare proprio i primi quattro bit di  $S'_B$  comunicandoli sul canale standard, l'intrusione viene scoperta.

1	2	3	4	5	6	7	...	successione temporale
1	0	1	1	1	0	0	...	$S_A$
+	x	+	x	x	+	x	...	basi di $A$
$\rightarrow$	/	$\rightarrow$	\	\	$\uparrow$	/	...	invio di $A$
+	+	+	x	+	x	+	...	basi di $E$
$\rightarrow$	$\rightarrow$	$\rightarrow$	\	$\uparrow$	/	$\rightarrow$	...	lettura e invio di $E$
1	1	1	1	0	0	1	...	$S_E$
+	x	+	+	+	x	x	...	basi di $B$
$\rightarrow$	/	$\rightarrow$	$\uparrow$	$\uparrow$	/	/	...	lettura di $B$
<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	...	$S_B$

## 23.2 BB84: Problemi di realizzazione e alternative

Il funzionamento visto è puramente esemplificativo, in quanto l'implementazione reale del protocollo è ben più complessa. In particolare, non è stato considerato un fattore estremamente importante: si può introdurre una certa quantità di **rumore** nella trasmissione dei fotoni, cosa che può portare Bob ad interpretare un fotone in modo sbagliato anche se avendo scelto la stessa base di Alice. Le probabilità che ciò accada sono naturalmente molto piccole, ma non trascurabili.

Bisogna quindi trovare il modo di capire quando le differenze nelle sequenze  $S'_B$  ed  $S'_A$  sono causate dal normale rumore di trasmissione e quando sono invece causate da un crittoanalista in ascolto sul canale quantistico.

### 23.2.1 Tasso di errore

In base alle caratteristiche del canale e dei dispositivi impiegati si stabilisce una percentuale prevedibile di bit errati dovuti al rumore, detta *Quantum bit error rate* (**QBER**), che viene prudenzialmente tenuta alta.

Prevedibilmente, se la percentuale di errori in  $S'_B$  supera il QBER, allora Alice e Bob concludono che vi è stata una intrusione di Eve o comunque un serio malfunzionamento del sistema e la chiave viene scartata.

Se non è stata rilevata un'intrusione ma c'è comunque qualche errore, Alice e Bob devono comunque ricostruire una chiave corretta. A tale scopo,  $S_A$  viene rappresentata usando un **codice a correzione di errore** commisurato al QBER. In tal modo Alice e Bob ricostruiscono la chiave corretta.

### 23.2.2 Ulteriori difese

Vi è comunque un attacco subdolo che Eve può intraprendere aggirando la misura del QBER. Eve intercetta solo una piccola parte della chiave con lo scopo di ottenere un'informazione parziale su di essa. Il numero di errori così introdotto è molto basso e il QBER può non essere superato.

Per tale motivo Alice e Bob, stabilita la sequenza corretta  $S_C$  derivante dall'informazione comune dopo la correzione data dal codice, eseguono una **amplificazione di privacy** dividendo  $S_C$  in blocchi e calcolando per ciascuno di essi una funzione hash crittografica la cui uscita sarà utilizzata come

chiave.

In questo modo Eve, che conosce solo alcuni bit di  $S_C$ , non potrà in alcun modo risalire ai bit della chiave.

### 23.2.3 Implementazione effettiva del BB84

Vediamo quindi adesso il reale protocollo BB84 per lo scambio quantistico di chiavi soggetto a interferenza di un crittoanalista attivo.

#### Sul canale quantistico

$S_A[1, n]$  è la sequenza iniziale di bit da cui estrarre la chiave, rappresentata con un codice a correzione di errore.

```
for i = 1 to n
```

1. **Alice** sceglie una base a caso, codifica  $S_A[i]$  ed invia il relativo fotone a Bob;
2. **Eve**, se presente, intercetta il fotone con una sua base, calcola  $S_E[i]$  e lo invia a Bob;
3. **Bob** sceglie una base a caso, interpreta il fotone ricevuto e costruisce  $S_B[i]$ .

#### Sul canale standard

QBER è il valore fissato come percentuale massima di bit errati dovuti a rumore;  $h$  è una funzione hash crittografica concordata.

1. **Bob** comunica ad Alice la sequenza di basi scelte;
2. **Alice** comunica a Bob le basi comuni;
3. **Alice e Bob** singolarmente e comunicando tra loro:
  - (a) estraggono le sottosequenze  $S'_A$  ed  $S'_B$  corrispondenti alle basi comuni, e due sottosequenze di esse  $S''_A$  ed  $S''_B$  in posizioni concordate;
  - (b) si scambiano  $S''_A$  ed  $S''_B$ ; se la percentuale di bit che differiscono è maggiore di QBER lo scambio è interrotto, altrimenti:
  - (c) calcolano le sequenze  $S'_A - S''_A$  ed  $S'_B - S''_B$ , e le codificano con il codice a correzione di errore ottenendo la sequenza comune  $S_C$ ;
  - (d) calcolano  $k = h(S_C)$ ; assumono quindi  $k$  come chiave.