Crittografia

Ahmad Shatti

2021-2022

Indice

Chapter 1

Introduzione

1.1 Crittografia

La crittografia è lo studio di tecniche matematiche sofisticate per mascherare i messaggi (**crittografia)** o tentare di svelarli (**crittoanalisi)**. L'insieme delle due discipline, crittografia e crittoanalisi, prende nome di **crittologia**, ovvero lo studio della comunicazione su canali non sicuri e relativi problemi. Scenario:

- Un agente *Alice* vuole comunicare con un agente *Bob*, e deve utilizzare un canale di trasmissione insicuro, cioè è possibile intercettare i messaggi che vi transitano
- Per proteggere la comunicazione i due agenti devono adottare un **metodo di cifratura**. *Alice* spedisce un messaggio in chiaro **m** sotto forma di testo cifrato (crittogramma) **c**, che deve essere:
  - incomprensibile al crittoanalista *Eve* (eavesdropper) in ascolto sul canale
  - facilmente decifrabile da *Bob*

In crittografia tutto quello che è legittimo (*Alice* e *Bob*) si deve poter fare in tempo polinomiale, invece, gli algoritmi a disposizione per chi non è il legittimo destinatario del messaggio (*Eve)* devono essere tipicamente esponenziali. La **cifratura del messaggio** è una funzione $\mathcal{C}$ iniettiva che ha come dominio lo spazio MSG dei testi in chiaro, come codominio lo spazio $CRITTO$ dei crittogrammi, e questa funzione associa ad un messaggio uno e un solo crittogramma $\mathcal{C} : MSG \to CRITTO$. Viceversa la funzione di **decifrazione** $\mathcal{D}$ è $\mathcal{D}$: $CRITTO \to MSG$. Le funzioni $\mathcal{C}$ e $\mathcal{D}$ sono una inversa dell'altra.

1.2 Livelli di segretezza

I cifrari sono classificati in base al livello di segretezza:

- **Cifrari per uso ristretto**, le funzioni di cifratura $\mathcal{C}$ e di decifrazione $\mathcal{D}$ sono tenute segrete in ogni loro aspetto. Impiegati per comunicazioni diplomatiche o militari. Non adatti per una crittografia "di massa";
- **Cifrari per uso generale**, ogni codice segreto non può essere mantenuto tale troppo a lungo. In un cifrario utilizzato da molti utenti, la parte segreta si limita alla **chiave k**, nota solo alla coppia di utenti che stanno comunicando. La chiave segreta k è diversa per ogni coppia di utenti e viene inserita come parametro nelle funzioni di cifratura e decifrazione. Le funzioni $\mathcal{C}$ e $\mathcal{D}$ sono pubbliche e solo la chiave deve essere segreta. Il vantaggio è che tenere segreta la chiave è più semplice che tenere segreto l'intero processo di cifratura e decifrazione, e tutti possono impiegare le funzioni pubbliche $\mathcal{C}$ e $\mathcal{D}$ a patto che scelgono chiavi diverse. Inoltre se un crittoanalista entra in possesso di una chiave occorre solo generarne una nuova, le funzioni $\mathcal{C}$ e $\mathcal{D}$ rimangono inalterate. Quando si progetta un cifrario lo si deve fare facendo in modo che esso sia sicuro anche se il nemico viene a sapere del sistema di cifratura adottato.

La segretezza dipende unicamente dalla chiave, quindi, il numero delle chiavi deve essere così grande da essere praticamente immune da ogni tentativo di provarle tutte. La chiave segreta deve essere scelta in modo casuale. Il crittoanalista potrebbe sferrare un attacco a forza bruta verificando la significatività delle sequenze $\mathcal{D}(c, k)$ per ogni k. Se $|Key| = 10^{20}$ e un calcolatore impiegasse 10^{-6} secondi per calcolare $\mathcal{D}(c, k)$ e verificarne la significatività occorrerebbe in media più di un milione di anni per scoprire il messaggio provando tutte le chiavi possibili. La segretezza può essere comunque violata con altre tecniche di crittoanalisi. Esistono cifrari più sicuri di altri, pur basandosi su uno spazio di chiavi molto più piccolo: un cifrario complicato non è necessariamente un cifrario sicuro; mai sottovalutare la bravura del crittoanalista

1.3 Attacchi a un sistema crittografico

Il comportamento di un crittoanalista può essere:

- **Passivo**, si limita ad ascoltare la comunicazione
- **Attivo**, agisce sul canale disturbando la comunicazione o modificando il contenuto dei messaggi

Gli attacchi a un sistema crittografico hanno l'obiettivo di **forzare** il sistema. Metodo e livello di pericolosità dipendono dalle informazioni in possesso del crittoanalista:

- **Cipher Text Attack** rileva sul canale una serie di crittogrammi
- **Known Plain-Text Attack** conosce una serie di coppie $(m_1, c_1), ..., (m_r, c_r)$ contenenti messaggi in chiaro e loro corrispondenti crittogrammi
- **Chosen Plain-Text Attack** si procura una serie di coppie $(m_1, c_1), ..., (m_r, c_r)$ relative a messaggi in chiaro da lui scelti
- **Chosen Cipher-Text Attack** si procura una serie di coppie $(m_1, c_1), ..., (m_r, c_r)$ relative a crittogrammi da lui scelti
- **Bruteforce Attack** si provano tutte le chiavi/password, etc...

Un cifrario si considera sicuro se non ci sono attacchi che sono meno costosi dell'attacco a forza bruta. Un altro tipo di attacco è il **Man in-the-middle** dove il crittoanalista si installa sul canale di comunicazione interrompendo le comunicazioni dirette tra i due utenti *Alice* e *Bob*, sostituisce i messaggi con altri messaggi propri e convince ciascun utente che tali messaggi provengono legittimamente dall'altro, ovvero *Eve* si finge *Bob* agli occhi di *Alice* e *Alice* agli occhi di *Bob*. L'attacco al sistema può essere portato a termine con

- **Successo pieno**, se si scopre la funzione $\mathcal{D}$
- **Successo limitato** si scopre solo qualche informazione su un particolare messaggio. L'informazione parziale può essere sufficiente per comprendere il significato del messaggio

1.4 Cifrari dichiarati sicuri

Esistono **cifrari perfetti** ovvero inattaccabili ma richiedono operazioni estremamente complesse e sono utilizzati solo in condizioni estreme. L'idea è che messaggio in chiaro e crittogramma risultano del tutto scorrelati tra loro, ovvero per il crittoanalista intercettare il crittogramma o no è la stessa cosa, nessuna informazione sul testo in chiaro può filtrare dal crittogramma. Un cifrario perfetto famoso è il **One-Time Pad** che è assolutamente sicuro ma richiede una chiave: lunga come il messaggio da scambiare, generata casualmente e la si può usare una volta sola. I cifrari diffusi al giorno d'oggi in pratica non sono perfetti, ma sono **dichiarati sicuri** perché rimasti inviolati dagli attacchi degli esperti e per violarli è necessario risolvere problemi matematici estremamente difficili. Il prezzo da pagare per forzare il cifrario è troppo alto perché valga la pena sostenerlo: l'operazione richiede di impiegare giganteschi calcolatori per tempi incredibilmente lunghi, quindi dal punto di vista pratico il cifrario è impossibile da forzare. Tuttavia non è noto se *la risoluzione di questi problemi matematici richieda* ***necessariamente*** *tempi enormi* oppure se *i tempi enormi di risoluzione siano dovuta alla nostra* ***incapacità di individuare metodi più efficienti*** di risoluzione. Attualmente quello che si usa nelle comunicazioni di massa è un cifrario simmetrico chiamato **AES**. È lo standard per le comunicazioni riservate ma "non classificate" cioè le comunicazioni in ambito privato commerciale. Questo cifrario è pubblicamente noto e realizzabile in hardware su computer di ogni tipo. Ha chiavi brevi (128 o 256 bit), il messaggio viene diviso in blocchi lunghi quanto la chiave e la stessa chiave è usata per cifrare e decifrare.

1.5 Chiave pubblica

La novità rispetto al passato è che le chiavi segrete non sono stabilite direttamente dai partner (*Alice* e *Bob*) ma dai mezzi elettronici che utilizzano per comunicare. Su Internet si costruisce una nuova chiave per ogni sessione. Il problema è come far scambiare la chiave in facilità e sicurezza: una intercettazione nell'operazione di scambio della chiave pregiudica il sistema. Nel 1976 viene proposto alla comunità scientifica un algoritmo per generare e scambiare una chiave segreta su un canale insicuro senza la necessità che le due parti si siano scambiate informazioni o incontrate in precedenza. Questo algoritmo detto **protocollo DH** è ancora largamente usato nei protocolli

crittografici su Internet. Nel 1976 Diffie e Hellman introdussero la **crittografia a chiave pubblica** con l'obiettivo di permettere a tutti di inviare messaggi cifrati ma di abilitare solo il destinatario a decifrarli. Nella crittografia a chiave pubblica le operazioni di cifratura e decifrazione utilizzano due chiavi diverse $k[pub]$ per cifrare e $k[prv]$ decifrare: la prima chiave è *pubblica*, cioè è nota a tutti, la seconda è *privata*, cioè nota soltanto al destinatario del messaggio. Come in precedenza le funzioni di cifratura $\mathcal{C}$ e decifrazione $\mathcal{D}$ sono di pubblico dominio, identiche per tutti gli utenti e si calcolano inserendovi una chiave come parametro; quindi la cifratura è accessibile, perché a tutti è nota la relativa chiave, mentre la decifrazione è accessibile solo a chi possiede la chiave privata. Per tale motivo i sistemi a chiave pubblica sono detti **asimmetrici**, mentre quelli a chiave segreta sono detti **simmetrici** perché la chiave di cifratura è uguale a quella di decifrazione (o l'una può essere facilmente calcolata dall'altra). Affinché sia possibile costruire un sistema a chiave pubblica la funzione $\mathcal{C}$ deve possedere una proprietà forte detta **one-way trap-door**, ovvero calcolare il crittogramma deve essere facile ma decifrazione deve essere computazionalmente difficile se non si conosce la chiave privata. Uno dei sistemi crittografici a chiave pubblica più importante è il cifrario **RSA**.

Vantaggi:

- *comunicazione molti a uno*: tutti gli utenti possono inviare in modo sicuro messaggi a uno stesso destinatario; solo il destinatario può cifrare i messaggi. Un crittoanalista non può ricavare informazioni sui messaggi pur conoscendo $\mathcal{C}, \mathcal{D}$ e $k[pub]$
- non è richiesto nessun scambio segreto di chiavi
- se gli utenti di un sistema sono n, il numero complessivo di chiavi (pubbliche e private) è $2n$ anziché $n(n-1)/2$

Svantaggi:

- sono molto più lenti di quelli basati sui cifrari simmetrici
- sono esposti ad attacchi di tipo *chosen plaintext*: si sceglie un numero qualsiasi di messaggi in chiaro, si costruisce i relativi crittogrammi e si ascolta sul canale confrontando i crittogrammi in transito e se si trova un riscontro si conosce esattamente qual è il messaggio passato

Si adotta solitamente un approccio *ibrido* che utilizza i cifrari simmetrici per la cifratura e decifrazione dei messaggi mediante chiavi segrete, e i cifrari simmetrici per lo scambio di tali chiavi. In questo modo la trasmissione dei messaggi lunghi avviene ad alta velocità, mentre è lento lo scambio delle chiavi segrete

Chapter 2

Rappresentazione e calcolabilità

2.1 Alfabeti e sequenze

Rappresentare matematicamente degli oggetti significa, fissato un **alfabeto** che contiene un numero finito di caratteri o simboli, associare ad ogni oggetto una sequenza ordinata ($ab \neq ba$) di caratteri dell'alfabeto che lo identifica in modo univoco. Rappresentare equivale ad "assegnare nomi" (le sequenze) ai singoli oggetti. La rappresentazione deve essere fatta in modo che oggetti diversi corrispondono sequenze diverse e il numero degli oggetti che si possono rappresentare non ha limiti. Si suppone che l'alfabeto Γ contenga s caratteri, in simboli $|\Gamma| = s$, e si vuole rappresentare N oggetti. Si indica con

- $d(s, N)$: lunghezza (numero di caratteri) della più lunga sequenza che rappresenta un oggetto dell'insieme
- $d_{min}(s, N)$: valore minimo di $d(s, N)$ tra tutte le rappresentazioni possibili

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

Esempio 1. $s = 1, \Gamma = \{0\}$, l'alfabeto contiene un solo carattere che è la cifra 0. Le sequenze di rappresentazione possono essere solo ripetizioni dello 0. Se si vuole costruire n sequenze diverse l'unica possibilità è farle di lunghezza crescente ($0, 00, 000, ...$), allora $d_{min}(1, N) = N$ che è una rappresentazione estremamente sfavorevole perché per rappresentare N oggetti si devono usare nomi che sono lunghi fino a N

Esempio 2. $s = 1, \Gamma = \{0, 1\}$, per ogni $k \geq 1$, 2^k sequenze di lunghezza k (*ogni carattere può essere scelto in due modi*). Il numero totale di sequenze lunghezze da 1 a k è dato da $\sum_{i=1}^{k} 2^i = 2^{k+1} - 2$ (*due sequenze di lunghezza uno:* 0, 1*; quattro sequenze di lunghezza due:* $00, 01, 10, 11$*; otto sequenze di lunghezza quattro...*). Per rappresentare N oggetti con queste sequenze deve risultare $2^{k+1} - 2 \geq N$, ovvero $k \geq log_2(N + 2) - 1$. Poiché $d_{min}(2, N)$ è pari al minimo intero k che soddisfa tale relazione si ha $d_{min}(2, n) = \lceil log_2(N + 2) - 1 \rceil$, da cui si ottiene:

$$\lceil log_2(N) \rceil - 1 \leq d_{min}(2, n) \leq \lceil log_2(N) \rceil$$

Dunque $\lceil log_2(N) \rceil$ caratteri binari sono sufficienti per costruire N sequenze differenti, e in particolari si possono costruire N sequenze differenti tutte di $\lceil log_2(N) \rceil$ caratteri. Per esempio si possono costruire le $N = 7$ sequenze: $0, 1, 00, 01, 10, 11, 000$ impiegando al massimo $\lceil log_2 7 \rceil = 3$ caratteri

$\lceil log_s N \rceil$ caratteri sono sufficienti per costruire N sequenze differenti, e in particolare si possono costruire N sequenze differenti tutte di $\lceil log_s N \rceil$ caratteri. Usare sequenze della stessa lunghezza è vantaggioso perché se si devono concatenare non si deve inserire un simbolo per marcare la separazione tra l'una e l'altra sequenza. Rappresentazioni che impiegano un numero massimo di caratteri in ordine logaritmico nella cardinalità N dell'insieme da rappresentare si dicono **efficienti**.

2.2 Teoria della calcolabilità

La calcolabilità è una branca dell'informatica teorica che si occupa delle questioni fondamentali sulla potenza i limiti dei sistemi di calcolo. I problemi oggetto di studio sono i problemi computazionali, ovvero, problemi formulati matematicamente di cui si cerca una soluzione algoritmica. La calcolabilità classifica i problemi computazionali in base al fatto che possono essere risolti o meno da un algoritmi: problemi **non decidibili** e **decidibili**. I problemi

che ammettono algoritmi di soluzione si dividono in problemi **trattabili** (costo polinomiale) e **intrattabili** (costo esponenziale). Un insieme è **numerabile** sse i suoi elementi possono essere messi in corrispondenza biunivoca con i suoi numeri naturali. Si considera l'insieme di tutte le sequenze generate con un alfabeto arbitrario e si cerca di elencarle in un ordine ragionevole. Poiché tali sequenze non sono in numero finito non si potrà completare l'elenco, ma lo scopo è quello di raggiungere qualsiasi σ sequenza arbitrariamente scelta in un numero finito di passi, cioè σ dovrà trovarsi a distanza finita dall'inizio dell'elenco. Per esempio si considerano le sequenze costruite sull'alfabeto $\Gamma = \{A, B, ..., Z\}$. Se come in un dizionario, si ponessero all'inizio tutte le sequenze che cominciano per A non si raggiungerebbe mai quelle che iniziano per B poiché le prime sono infinite. Si ricorre quindi a un **ordinamento canonico** costruito come:

- si stabilisce un ordine tra i caratteri dell'alfabeto
- si ordinano le sequenze in ordine di lunghezza crescente e, a pari lunghezza in ordine alfabetico

Nell'esempio precedente si avrà:

$$A,\ B,\ C,\ ...,Z$$
$$AA,\ AB,\ ...,\ AZ,\ BA,\ ...,\ BZ,\ ...,\ ZA,\ ...,\ ZZ$$
$$AAAA,\ ...$$

Ad una sequenza arbitraria corrisponde il numero che ne indica la posizione e, a un numero naturale n corrisponde la sequenza che occupa la posizione n nell'elenco. La numerazione delle sequenze è possibile perché esse sono di lunghezza finita anche se illimitata. L'ordinamento canonico permette di dimostrare che le sequenze sono numerabili. L'insieme dei problemi computazioni non è numerabile. Un problema computazionale può essere visto come una funzione matematica che associa ad ogni insieme di dati, espressi da k numeri interi, il corrispondente risultato, espresso da j numeri interi $f : N^k \to N^j$. Si dimostra che non è possibile costruire un elenco di funzioni e lo si fa per un caso particolare: $f : N \to \{0, 1\}$. Ogni $f \in F$ può essere rappresentato da una sequenza infinita:

x	0	1	2	3	4	...	n	...
$f(x)$	0	1	0	1	0	...	0	...

o se è possibile, da una regola finita di costruzione $f(x) = \begin{cases} 0 & x\ pari \\ 1 & x\ dispari \end{cases}$

x	0	1	2	3	4	5	6	7	8	...
$f_0(x)$	1	0	1	0	1	0	0	0	1	...
$f_1(x)$	0	0	1	1	0	0	1	1	0	...
$f_2(x)$	1	1	0	1	0	1	0	0	1	...
$f_3(x)$	0	1	1	0	1	0	1	1	1	...
$f_4(x)$	1	1	0	0	1	0	0	0	1	...
...				...					...	

Per assurdo si ipotizza F numerabile allora si può assegnare ad ogni funzione un numero progressivo nella numerazione e costruire una tabella infinita con tutte le funzioni. Adesso l'obiettivo della dimostrazione è costruire una funzione $g(x)$ e far vedere che rimane esclusa dalla tabella. Si definisce $g(x)$ come il complemento del valore di $f_x(x)$

$$g(x) = \begin{cases} 0 & f_x(x) = 1 \\ 1 & f_x(x) = 0 \end{cases} \quad \Rightarrow\ g \text{ non corrisponde ad alcuna delle } f_i \text{ della tabella perché differisce da tutte le funzioni almeno nella diagonale principale}$$

Per qualunque numerazione scelta esiste sempre almeno una funzione esclusa, quindi F non è numerabile. Esistono infinite funzioni di F escluse da qualsiasi numerazione. L'insieme dei problemi computazionali non è numerabile

2.2.1 Problema dell'arresto

L'informatica rappresenta tutte le sue entità in forma digitale come sequenze finite di simboli di alfabeti finiti. Un algoritmo è una sequenza finita di operazioni, completamente e univocamente determinate. La formulazione di un algoritmo dipende dal modello di calcolo utilizzato, ma qualsiasi modello si scelga, gli algoritmi devono essere descritti, ossia rappresentati da sequenze finite di caratteri di un alfabeto finito. Gli algoritmi sono possibilmente infiniti ma numerabili. I problemi computazionali, come visto prima, non sono numerabili quindi esistono dei problemi privi di un corrispondente algoritmo di calcolo. Turing ha verificato che riuscire a dimostrare se un programma arbitrario si arresta e termina in tempo finito la sua esecuzione è in generale impossibile. Per esempio **il problema dell'arresto**: è un problema posto in forma decisionale (ovvero problemi la cui risposta può essere solo *True/False*) ed è formulato in questo modo: *preso un algoritmo arbitrario A e un insieme di dati di input D, decidere in **tempo finito** se la computazione di A su D termina o no* (supponendo di non avere limiti di tempo e memoria). *Dimostrazione:* se il problema dell'arresto fosse decidibile, allora esisterebbe un algoritmo *arresto* che, presi A e D come dati in input, risponde in tempo finito `ARRESTO(`A, D`) = 1` se $A(D)$ termina, altrimenti, `ARRESTO(`A, D`) = 0` se $A(D)$ non termina. Si può scegliere $D = A$, cioè considerare la computazione $A(A)$. Se esistesse l'algoritmo ARRESTO, esisterebbe l'algoritmo `PARADOSSO(A)` che prende in input un algoritmo generico A:

```
PARADOSSO(A)
while(ARRESTO(A, A)){
    ; }
```

consiste in un ciclo la cui guardia è la chiamata dell'algoritmo ARRESTO e il corpo del while non ha nessuna operazione. Se la guardia è vera, l'algoritmo non termina, ed essa è vera quando `ARRESTO`(A, A) `= 1`. Se la guardia è falsa, PARADOSSO termina immediatamente. Se si calcola `PARADOSSO(PARADOSSO)` questo termina se e solo se `ARRESTO(PARADOSSO, PARADOSSO) = 0` e ciò è equivalente a dire che `PARADOSSO(PARADOSSO)` non termina, quindi si ha una contraddizione. L'unico modo di risolvere la contraddizione è che l'algoritmo PARADOSSO non possa esistere, dunque non può esistere nemmeno l'algoritmo ARRESTO, e questo dimostra che il problema dell'arresto è indecidibile. La teoria della calcolabilità dipende dal modello di calcolo? I linguaggi di programmazione esistenti sono tutti equivalenti? È possibile che problemi oggi irrisolvibili possano essere risolti in futuro con altri linguaggi o calcolatori? Queste sono tutte domande per le quali non si conosce una risposta certa o una risposta sotto forma di teorema, ma si conosce la **tesi di Church-Turing**: tutti i ragionevoli modelli di calcolo risolvono esattamente la stessa classe di problemi e dunque si equivalgono nella possibilità di risolvere problemi, pur operando con diversa efficienza. Questa tesi porta a dire che la decidibilità è una proprietà del problema, e quindi non dipende dal modello di calcolo, incrementi qualitativi alla struttura di una macchina servono solo a abbassare il tempo d'esecuzione e/o rendere più agevole la programmazione

2.3 Classi di complessità

Esistono problemi (*clique, cammino hamiltoniano, zaino, ...*) il cui stato computazionale non è chiaro, ovvero si hanno a disposizione solo algoritmi di costo esponenziale e non si hanno limiti inferiori per dimostrare che non possono esistere algoritmi di costo polinomiale. Questi problemi si chiamano **problemi presumibilmente intrattabili**. Dato un problema decisionale Π ed un algoritmo A, si dice che A risolve Π se, data un'istanza di input x, $A(x) = 1 \Leftrightarrow \Pi(x) = 1$, A risolve Π in tempo $t(n)$ e spazio $s(n)$ se il tempo di esecuzione e l'occupazione di memoria di A sono rispettivamente $t(n)$ e $s(n)$. Data una qualunque funzione $f(n)$ si definiscono:

- **Classe $\boldsymbol{Time(f(n))}$**: insieme dei problemi decisionali che possono essere risolti in tempo $O(f(n))$
- **Classe $\boldsymbol{Space(f(n))}$**: insieme dei problemi decisionali che possono essere risolti in spazio $O(f(n))$

e inoltre:

- **Classe P**: classe dei problemi che ammettono algoritmi di risoluzione polinomiali
- **Classe PSPACE**: classe dei problemi risolvibili in spazio polinomiale
- **Classe EXPTIME**: classe dei problemi risolvibili in tempo esponenziale

dove P $\subseteq$ PSpace $\subseteq$ ExpTime. Non è noto (ad oggi) se le inclusioni siano proprie. L'unico risultato di separazione dimostrato finora riguarda P e ExpTime. Alcuni problemi interessanti:

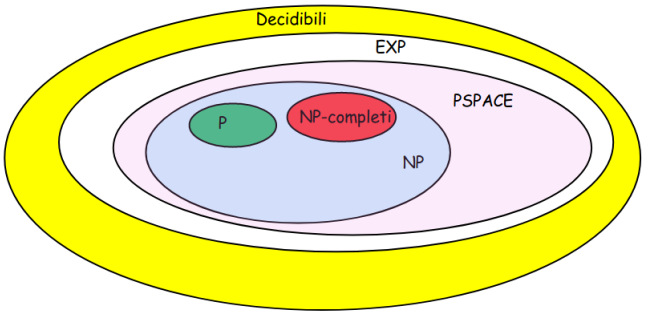
- **Clique**, si considerano tutti i sottoinsiemi di vertici in ordine di cardinalità decrescente, e si verifica se formano una clique di dimensione almeno k. L'unica soluzione conosciuta è quello a forza bruta in tempo esponenziale, quindi clique $\in$ ExpTime
- **Cammino Hamiltoniano**, si considerano tutte le permutazioni di vertici, e si verifica se i vertici in quell'ordine sono a due adiacenti. L'unica soluzione conosciuta è $n!$ dove n è il numero di vertici. Cammino Hamiltoniano $\in$ ExpTime
- **SAT**, siano V un insieme di variabili booleane: si riferisce ad una variabile o alla sua negazione come un *letterale*. Se si fa l'OR di un gruppo di letterali si crea una *clausola*. Un espressione booleana su V si dice *forma normale congiuntiva* se è espressa come congiunzione di clausole, ovvero come un AND di clausole ognuna formata da un OR di letterali. Data un espressione in forma normale congiuntiva, il problema SAT chiede di verificare se esiste un qualche assegnamento di valore *true* o *false* tali da rendere l'intera espressione vera. L'unica soluzione conosciuta finora è quella a forza bruta. SAT $\in$ ExpTime

Per tutti questi problemi non si conosce il limite inferiore, cioè non si sa se esiste un algoritmo più efficiente. Per alcuni problemi, per le istanze accettabili è possibile fornire un **certificato**, cioè delle informazioni aggiuntive che permettono di verificare se l'istanza ha la proprietà cercata in tempo polinomiale. In poche parole è come se si avesse già la soluzione che soddisfa una certa proprietà di un determinato problema e la si debba solo controllare, questo procedimento lo si può fare in tempo polinomiale. I certificati si definiscono solo per le istanze accettabili

ed in generale non è facile costruire certificati di non esistenza. Chi non ha un certificato, può individuarla in tempo esponenziale considerando tutti i casi possibili con il metodo della forza bruta. Un certificato per il problema clique è un sottoinsieme di k vertici che forma la clique. Un problema Π è **verificabile in tempo polinomiale** se:

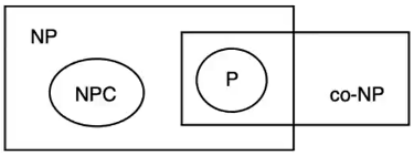
- ogni istanza accettabile x di Π di lunghezza n ammette un certificato y di lunghezza polinomiale in n
- esiste un algoritmo di verifica polinomiale in n e applicabile a ogni coppia $< x, y >$ che permette di attestare che x è accettabile

La **classe NP** è la classe dei problemi decisionali verificabili in tempo polinomiale dove la *N* sta per tempo polinomiale *non deterministico*. Naturalmente la classe P è incluso in NP perché ogni problema in P ammette un certificato verificabile in tempo polinomiale: si esegue l'algoritmo che risolve il problema per costruire il certificato. Le due classi P e NP sono uguali o diverse? In altre parole risolvere e verificare sono la stessa cosa? Si pensa che sono diverse ma non lo si è dimostrato. Con la congettura $P \neq NP$, si cerca di individuare i problemi più difficili all'interno della classe NP che prendono il nome di **problemi NP-completi**. Se esistesse un algoritmo polinomiale per risolvere uno solo di questi problemi, allora tutti i problemi in NP potrebbero essere risolti in tempo polinomiale, e dunque $P = NP$. Per definire i problemi NP-completi si ha bisogno della **riduzione polinomiale**, ovvero si converte un problema in un altro: dati due problemi decisionali Π_1 e Π_2, e siano I_1 e I_2 insiemi delle istanze di input di Π_1 e Π_2, si dice che Π_1 si riduce in tempo polinomiale a Π_2 (notazione $\Pi_1 \leq_p \Pi_2$) se esiste una funzione $f : I_1 \to I_2$ calcolabile in tempo polinomiale tale che, per ogni istanza x di Π_1, x è un'istanza accettabile di Π_1 se e solo se $f(x)$ è una istanza accettabile di Π_2. In altre parole in tempo polinomiale si vuole trasformare il problema Π_1 nel problema Π_2 in modo tale che x sia un'istanza accettabile per Π_1 se e solo se la sua traduzione $f(x)$ in istanza per il problema Π_2 è accettabile per Π_2. Il concetto di riduzione polinomiale ha la conseguenza se esistesse un algoritmo per risolvere Π_2 lo si potrebbe utilizzare per risolvere Π_1. Un problema Π si dice **NP-hard** se per ogni $\Pi' \in NP$ lo si può ridurre in tempo polinomiale a Π. Dimostrare che un problema è in NP può essere facile, basta esibire un certificato polinomiale. Non è altrettanto facile dimostrare che un problema Π è NP-completo o NP-hard: si dovrebbe dimostrare che tutti i problemi in NP si riducono polinomialmente a Π. In realtà la prima dimostrazione di NP-completezza "aggira" il problema: il **teorema di cook** dimostra che SAT è NP-completo affermando che dati un qualunque problema Π in NP ed una qualunque istanza x per Π, si può costruire una espressione booleana in forma normale congiuntiva che descrive il calcolo di un algoritmo per risolvere Π su x, l'espressione è vera se e solo se l'algoritmo restituisce 1. La riduzione polinomiale è transitiva, quindi per dimostrare che un problema Π è NP-completo: per ogni generico problema $\Pi' \in NP$; si riduce il generico problema a SAT perché esso è NP-completo: $\Pi' \leq_p$ SAT; SAT si riduce a Π se si è riusciti a costruire la riduzione: SAT $\leq_p \Pi$ e dunque per transitività $\Pi' \leq_p \Pi$. Tutti i problemi NP-completi sono tra loro NP equivalenti. Sono tutti facili, o tutti difficili.



2.3.1 Classi co-P e co-NP

Un problema complementare accetta tutte e sole le istanze rifiutate dal problema di partenza. Dal punto di vista della complessità vi è una profonda differenza tra il certificare l'esistenza o la non-esistenza di una soluzione, a seconda se il problema sta in P oppure NP. Se il problema è nella classe P allora anche il complementare lo sarà (si risolve il problema di partenza se restituisce 1, allora il complementare risponderà 0). Quindi si può dire che P = co-P. Questo non vale per la classe co-NP, si congettura che NP e co-NP siano diverse. Se questa congettura è vera, allora P $\neq$ NP



Chapter 3

Il ruolo del caso

3.1 Significato algoritmo della casualità

La crittografia utilizza delle **sequenze casuali** per due importanti motivi: per alimentare gli algoritmi randomizzati e per generare le chiavi segrete dei cifrari. Date due sequenze si mette a confronto una sequenza composta solo di uni con un'altra in cui non si vede una regola precisa di generazione:

$$h_1 = 11111111111111...1 \qquad |h_1| = 20$$
$$h_2 = 01100101110100...0 \qquad |h_2| = 20$$

Per ipotesi h_1 e h_2 sono state generate con i lanci di una moneta. La probabilità di ottenere h_1 è $P(h_1) = (1/2)^{20}$. Stessa cosa per $P(h_2)$. Entrambe le probabilità sono molto piccole ma tuttavia esaminando le due sequenze si è indotti a credere che h_1 non sia casuale perché ha una "regola" che permette di descriverla in modo semplice come "sequenza composta di 20 uni". L'idea è che una sequenza binaria h è casuale se non ammette un algoritmo di generazione A la cui rappresentazione binaria è più corta della sequenza h. Si suppone adesso che la rappresentazione h consiste di n uni e si definisce un algoritmo A che realizzi l'operazione "genera n uni":

```
for(i = 0; i < n; i++) print '1';
```

quindi h sarà lungo n, mentre la lunghezza di A sarà $c + \Theta(logn)$. Quindi con ordine di $logn$ bit si è descritto una sequenza lunga n. Se h appare casuale, o non presenta evidenti regolarità, allora l'algoritmo di generazione conterrà la sequenza stessa al suo interno e la restituirà in output. In questo caso il programma costerà $c + \Theta(n)$

3.1.1 Formalizzazione matematica

Quando si parla di algoritmi si deve specificare il modello di calcolo in cui gli algoritmi verranno eseguiti. Si vuole rendere la definizione di casualità indipendente dal sistema di calcolo adottato. I sistemi di calcolo sono tantissimi ma numerabili (perché si è in grado di descriverli). Si definisce la **complessità di Kolmogorov** della sequenza h rispetto al sistema di calcolo S_i come: $\mathcal{K}_{S_i}(h) = min\{|p| : S_i(p) = h\}$, ovvero la lunghezza del programma più breve in grado di generare la sequenza h nel sistema S_i. Tra i vari sistemi di calcolo esiste almeno un S_u **sistema di calcolo universale** che può simulare il funzionamento di tutti gli altri sistemi di calcolo se gli si fornisce la coppia $q = \langle i, p \rangle$, dove i è l'indice del sistema S_i da simulare e p è il programma di questo. La simulazione genera la stessa sequenza h, ovvero $h = S_i(p) = S_u(\langle i, p \rangle)$. Il programma $q = \langle i, p \rangle$ per S_u ha lunghezza $|q| = |i| + |p| = \Theta(log_2 i) + |p|$ dove $\Theta(log_i)$ non dipende da h. Quindi per qualsiasi h si ha $\mathcal{K}_{S_u}(h) \leq \mathcal{K}_{S_i}(h) + c_i$ con c_i costante positiva che dipende solo dal sistema S_i di riferimento. La complessità secondo Kolmogorov di una sequenza h è $\mathcal{K}(h) = \mathcal{K}_{S_u}(h)$, infatti $\forall S_i$ la complessità $\mathcal{K}_{S_i}(h)$ è maggiore uguale a $\mathcal{K}_{S_u}(h)$ a meno di una costante.

3.1.2 Sequenze casuali

Una sequenza h è casuale se $\mathcal{K}(h) \geq |h| - \lceil log_2 h \rceil$, la casualità è una proprietà della sequenza stessa, cioè è indipendente dal modo in cui la sequenza è ottenuta.

Dimostrazione. *Esistenza di sequenze casuali.*
Si indica con $S = \#$ sequenze binarie lunghe $n = 2^n$ e con $T = \#$ sequenze binarie lunghe n non casuali. L'obiettivo è dimostrare $T < S$. Le T sequenze non casuali sono generate da programmi lunghe strettamente minore di $n - \lceil log_2 n \rceil$. Si pone con: $N = \#$ sequenze binarie lunghe $< n -$

$\lceil log_2 n \rceil = \sum_{i=0}^{n-\lceil logn \rceil - 1} 2^i = 2^{n-\lceil logn \rceil - 1}$. Tra queste N sequenze ci sono anche i programmi che generano le T sequenze non casuali di lunghezza n allora $T \leq N < S$ e quindi $T < S$

Le sequenze casuali esistono e sono enormemente più numerose delle sequenze non casuali. Il problema di stabilire se una sequenza arbitraria è casuale è **indecidibile**

Dimostrazione. Per assurdo si suppone che esiste un algoritmo $RANDOM(h) = \begin{cases} 1 & se\ h\ casuale \\ 0 & altrimenti \end{cases}$

Si costruisce l'algoritmo PARADOSSO che enumera tutte le possibili sequenze binarie in ordine crescente di lunghezza

```
PARADOSSO()
for(binary h = 1 to ∞)
    if(|h| - ⌈ log|h|⌉ > |P| && RANDOM(h) == 1)
        return h
```

dove il *for* genera le sequenze binarie in ordine di lunghezza crescente e l'*if* controlla se la sequenza è abbastanza lunga e se la sequenza è casuale. PARADOSSO restituisce la prima sequenza casuale di lunghezza $|h|$ tale che $|h| - \lceil log|h| \rceil > |P|$. P è una sequenza binaria che rappresenta la codifica del programma complessivo PARADOSSO + RANDOM, ovvero $|P| = |\text{PARADOSSO}| + |\text{RANDOM}|$, ed è una costante che non dipende da h perché h non compare in P ma solo come nome di variabile. La prima condizione dell'*if* chiede $|h| - \lceil log|h| \rceil > |P|$ ma P è un programma che genera h, ed è un programma breve perché la sua lunghezza è minore di $|h| - \lceil log|h| \rceil$. Questa condizione è equivalente a dire che h non è casuale. La seconda condizione dice che h è casuale, quindi il programma termina quando la sequenza h è allo stesso tempo non casuale e casuale. Dunque il programma PARADOSSO non può esistere ma lo si è costruito in modo del tutto legittimo $\Rightarrow$ *contraddizione*. Significa che la premessa è falsa ovvero l'esistenza dell'algoritmo RANDOM

3.2 Generatori di numeri pseudo-casuali

Nel caso binario una sorgente casuale genera una sequenza di bit che hanno queste due proprietà:

1. $P(0) = P(1) = \frac{1}{2}$, ovvero 0 e 1 si presentano con pari probabilità. Questa proprietà non è così rigida quindi la si può sostituire con $P(0) > 0, P(1) > 0$ immutabili durante il processo. Si può sostituire la proprietà perché se $P(0) > P(1)$ allora è **sempre possibile bilanciare la sequenza**

   Esempio. Si genera la sequenza 01100010001011100001001, si dividono in coppie: $01\ 10\ 00\ 10\ 01\ 01\ 11\ 00\ 00\ 10\ 01$, si eliminano le coppie dove i bit sono uguali: $01\ 10\ 10\ 01\ 01\ 10\ 01$, dopo di che si associano le coppie miste ovvero $01 \rightarrow 0$ e $10 \rightarrow 1$. Siccome le coppie di bit diverse compaiono con pari probabilità, la sequenza la si può riscrivere come: 0110010. Questa è una tecnica che permette data una sorgente binaria casuale sbilanciata di renderla bilanciata

2. la generazione di un bit è indipendente dalla generazione dei bit precedenti, ovvero, non si può prevedere il valore di un bit osservando quelli già generati

Benché le sequenze casuali esistono e, come si è dimostrato, sono in larga maggioranza, non è semplice individuare sorgenti casuali, in particolare è difficile garantire che la generazione di un bit avvenga in modo indipendente dalla generazione degli altri, poiché nella realtà ogni esperimento modifica l'ambiente circostante e può influenzare l'esperimento successivo. Per la generazione di sequenze *brevi* si sfruttano le casualità presenti in processi fisici, come *valori campionati del rumore proveniente da un microfono*, o processi software come *valori casuali dello stato dell'orologio interno di un calcolatore o la posizione della testina su un disco rigido*. I due approcci producono risultati ragionevolmente imprevedibili se non si ha accesso fisico ai dispositivi utilizzati, i quali altrimenti potrebbero essere persino manomessi falsando il processo di generazione. Il loro impiego è però problematico per la difficoltà di realizzazione. Nella pratica la perfetta casualità di una sorgente non potrà essere garantita e quindi si dovranno accettare alcuni compromessi. Il numero di bit casuali richiesti nelle applicazioni crittografiche può essere molto elevato e quindi si ricorre in genere a un diverso meccanismo algoritmico che ricerca la casualità all'interno di alcuni processi matematici. Un **generatore di numeri pseudo-casuali** è un algoritmo che prende in input un valore iniziale detto **seme** e genera in output una sequenza arbitrariamente lunga di numeri che all'interno contiene una sottosequenza detta **periodo** che si ripete indefinitamente. Un generatore è tanto migliore quanto più lungo è il periodo. Siccome il processo è deterministico se si innesca il generatore dandogli in input lo stesso seme, si ottiene esattamente la stessa sequenza. Sia il seme $|S| = s$ bit, il generatore di numeri pseudo-casuali genera al più 2^s sequenze diverse

Generatore lineare: $x_i = (a \cdot x_{i-1} + b) \; mod \; m$ con $a, b, m \in \mathbb{N}$ parametri del generatore

Il periodo è al massimo m, se i parametri sono scelti accuratamente, il generatore lineare produce una permutazione degli interi da $[0, m-1]$. Per garantire che un generatore produca effettivamente una permutazione esistono dei criteri: $mcd(b, m) = 1$, $(a-1)$ deve essere divisibile per ogni fattore primo di m e se 4 divide m allora deve dividere anche $(a-1)$. Valori consigliati sono per esempio $a = 3141592653, b = 2718281829, m = 2^{32}$

Generatore polinomiale: $x_i = (a_1 x_{i-1}^t + a_2 x_{i-1}^{t-1} + ... + a_t x_{i-1} + a_{t+1}) \; mod \; m$

Questi generatori possono essere facilmente trasformati per produrre sequenze binarie pseudo-casuali: si calcola il valore $r = x_i/m$ se la prima cifra decimale di r è dispari il bit generato è uno, altrimenti è zero. I generatori lineari e polinomiali sono particolarmente efficienti ma non impediscono di fare previsioni sugli elementi generati, neanche quando il seme impiegato è strettamente casuale. Esistono infatti algoritmi che permettono di scoprire in tempo polinomiale i parametri del generatore partendo dall'osservazione di alcune sequenze prodotte, e questo ne svela completamente il funzionamento. Affinché questa condizione non sia verificata occorre rafforzare definizioni e test.

3.2.1 Test statistici

I test standard di valutazione verificano che le sequenze prodotte presentano le proprietà tipiche di una sequenza casuale:

- **Test di frequenza**, verifica se i diversi elementi appaiono in S approssimativamente lo stesso numero di volte
- **Poker test**, verifica la equa distribuzione di sottosequenze di lunghezza arbitraria ma prefissata
- **Test di autocorrelazione**, verifica il numero di elementi ripetuti a distanza prefissata
- **Run test**, verifica se le sottosequenze massimali contenenti elementi tutti uguali hanno una distribuzione esponenziale negativa, cioè più sono lunghe meno sono frequenti

Per le applicazioni crittografiche si richiede anche il **test di prossimo bit** che verifica che non esiste un algoritmo polinomiale in grado di prevedere l'$i+1$-esimo bit generato a partire dalla conoscenza degli $i-$bit precedentemente generati con probabilità maggiore di $\frac{1}{2}$. Se il generatore supera il test di prossimo bit allora supera anche gli altri quattro test statistici e si dice **crittograficamente sicuro**

3.2.2 Generatori crittograficamente sicuri

Per costruire generatori crittograficamente sicuri si ricorre alle **funzioni f one-way** che sono computazionalmente facili da calcolare e difficili da invertire, cioè si conosce un algoritmo polinomiale per il calcolo di $y = f(x)$ ma si conoscono solo algoritmi esponenziali per il calcolo di $x = f^{-1}(y)$. Idea: con f one-way si sceglie un seme x_0, si ottiene $x_1 = f(x_0)$, poi $x_2 = f(x_1) = f(f(x_0))$, e così via, cioè si itera l'applicazione della funzione one-way un numero arbitrario di volte. Ogni elemento della S si può calcolare efficientemente dal precedente, ma non dai successivi perché f è one-way. Se dunque si calcola S per un certo numero di passi senza svelare il risultato, e si comunicano poi gli elementi uno dopo l'altro in **ordine inverso**, ciascun elemento non è prevedibile in tempo polinomiale pur conoscendo quelli comunicati prima di esso. Nelle applicazioni crittografiche si impiegano generatori binari ed essi vengono costruiti impiegando particolari **predicati** delle funzioni one-way, cioè proprietà che possono essere vere o false per ogni valore di x. Un predicato $b(x)$ è detto **hard-core** per una funzione one-way $f(x)$ se $b(x)$ è facile da calcolare conoscendo il valore di x, ma è difficile da calcolare conoscendo solo il valore di $f(x)$

Esempio. $f(x) = x^2 \; mod \; n$ se n non è primo.

Posto per esempio $n = 77$ e $x = 10$ è polinomialmente facile calcolare il valore $y = 10^2 \; mod \; 77 = 23$ ma è esponenzialmente difficile risalire al valore di $x = 10$ dato $y = 23$. Questa operazione infatti si sa eseguire solo attraverso un numero esponenziale di tentativi. Ora il predicato $b(x) = "x \; è \; dispari"$ è hard-core, infatti il valore di x permette di calcolare immediatamente $b(x) = 1$ se x è dispari, $b(x) = 0$ se x è pari, ma il problema di calcolare $b(x)$ conoscendo solo il valore $y = x^2 \; mod \; n$ è esponenzialmente difficile

Il **generatore BBS** è un generatore crittograficamente sicuro e funziona in questo modo: sia $n = p \cdot q$ il prodotto di due numeri primi grandi tali che:

- $p \; mod \; 4 = 3$ e $q \; mod \; 4 = 3$
- $2 \cdot \lfloor p/4 \rfloor + 1$ e $2 \cdot \lfloor q/4 \rfloor + 1$ siano coprimi

dopodiché si sceglie un y coprimo con n, e si calcola $x_0 = y^2 \ mod \ n$. Il generatore BBS impiega x_0 come seme, calcola una successione x_i di $m \leq n$ interi e genera in corrispondenza una sequenza binaria b_i secondo la legge: $x_i = (x_{i-1})^2 \ mod \ n, \ b_i = 1 \Leftrightarrow x_{m-1}$ è dispari. Il generatore parte da x_0 cui corrisponde b_m, e procede al calcolo degli interi $x_1, ..., x_m$ memorizzando i corrispondenti bit $b_{m-1}, ..., b_0$ che comunica poi all'esterno in ordine inverso

Esempio. Si sceglie $p = 11$ e $q = 19$
Si controlla: $11 \ mod \ 4 = 3$ e $19 \ mod \ 4 = 3 \Rightarrow$ OK; poi $2 \cdot \lfloor 11/4 \rfloor + 1 = 5$ e $2 \cdot \lfloor 19/4 \rfloor + 1 = 9 \Rightarrow$ OK
Si lavora con $n = 11 \cdot 19 = 209$, si sceglie $y = 30$ che è coprimo con 209

ora si calcola:
$x_0 = 30^2 \ mod \ 209 = 64$
$x_1 = 64^2 \ mod \ 209 = 125$
$x_2 = 125^2 \ mod \ 209 = 159$
$x_3 = 159^2 \ mod \ 209 = 201$
...

ora si calcola la parità:

x_0	x_1	x_2	x_3	...
64	125	159	201	
0	**1**	**1**	**1**	

⟵ *si consuma in questo senso*

Il generatore BBS è piuttosto lento poiché il calcolo di ogni bit richiede l'esecuzione di un elevamento al quadrato di un numero di grandi dimensioni. Essi sono impiegati nella creazione di chiavi segrete corte, ma non quando si richiedono lunghe sequenze casuali: poiché questo caso, si preferiscono generatori teoricamente meno sicuri (comunque a tutt'oggi inviolati) ma efficientissimi e si basano sui *cifrari simmetrici*. Si usa un cifrario simmetrico (*DES, 3DES, AES*) e siano r il numero di bit delle parole prodotte, s un seme casuale di r bit, m il numero di parole che si desidera produrre e k la chiave segreta del cifrario. Algoritmo:

```
GENERATORE(s, m)
d = "rappresentazione su r bit di data e ora attuale";
y = C(d, k);
z = s
for (i = 1 to m)
    x_i = C(y XOR z, k);
    z = C(y XOR x_i, k);
    comunica all'esterno x_i
```

3.3 Algoritmi randomizzati

Una utilizzazione delle sorgenti casuali all'interno di un algoritmo è volta ad aggirare alcune situazioni sfavorevoli in cui si possono presentare i dati, combinando questi con una sequenza casuale. L'esempio più noto è il **Quick sort** che anziché operare sui dati d'ingresso nell'ordine in cui appaiono, l'algoritmo ne costruisce una permutazione casuale e opera su questa. Gli algoritmi randomizzati si dividono in due classi:

- **Las Vegas**, generano un risultato *sicuramente* corretto in tempo *probabilmente* breve.
  Esempio tipico: Quick sort
- **Monte Carlo**, generano un risultato *probabilmente* corretto in tempo *sicuramente* breve
  La probabilità di errore deve essere matematicamente misurabile e arbitrariamente piccola
  Esempio tipico: test di primalità

Si illustra il problema $\mathcal{P}_{primo}(N)$ che chiede di stabilire se un dato numero N è primo. La soluzione di $\mathcal{P}_{primo}(N)$ è importante in molti protocolli crittografici. Il primo algoritmo polinomiale per la sua soluzione aveva una complessità di $O(n^{12})$ successivamente ridotta fino a $O(n^6)$, valore ancora estremamente alto. Si deve quindi cercare un algoritmo efficiente per risolvere il problema pur correndo il rischio di dare una risposta errata, purché la probabilità che ciò avvenga sia trascurabile. Si riporta, per la sua semplicità ed efficienza, il **test di primalità di Miller - Rabin**.

3.3.1 Test di primalità di Miller - Rabin

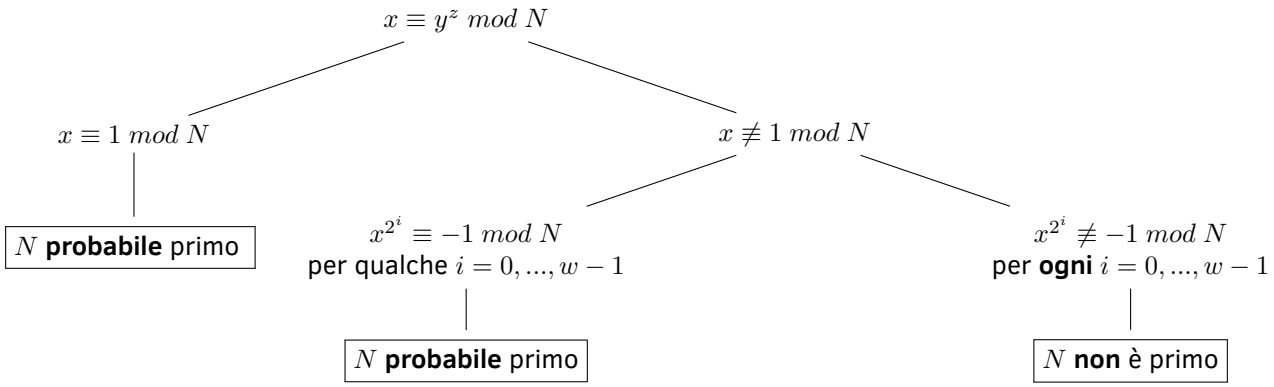
Si basa su sia N dispari (altrimenti è certamente composto), e si pone $N - 1 = 2^w z$ con z dispari. Sia y un numero arbitrario detto **testimone** tale che $1 < y < N$. Se y è coprimo con N allora vale una delle due seguenti:

i) $y^z \equiv 1 \ mod \ N$

ii) $y^{2^i z} \equiv -1 \ mod \ N$ per qualche $i = 0, ..., w - 1$

Applicazione:

1. N candidato primo dispari
2. Si scrive $N-1=2^w z$ con z dispari
3. Si sceglie y tra $1<y<N$ e si calcola:



Il test di Miller-Rabbin presenta il vantaggio che permette di calcolare nel caso di "N probabile primo" qual è la probabilità di errore. Si dimostra infatti che questa probabilità è minore di $1/4$ dopo una iterazione. Questa probabilità si può rendere più piccola se si effettuano più iterazioni del test, ovvero, ogni volta che si ottiene come esito "N probabile primo", si ripete il test scegliendo un diverso valore di y allora la probabilità di errore scende a $(1/4)^k$ per ogni k volte si è provato il test.

Esempio. Si prende come candidato primo $N=97$ che si riscrive come $N-1=96=2^5\cdot 3$. Si sceglie un y compreso tra 1 e 97, per esempio $y=2$ e si calcola $x=2^3 \ mod \ 97 = 8 \mod 97$. Quindi si è nel caso $x \not\equiv 1 \ mod \ N$, allora si controlla gli i compresi tra 0 e 4:

$$x = 8^{2^0} \ mod \ 97 = 8 \ mod \ 97$$
$$x = 8^{2^1} \ mod \ 97 = 64 \ mod \ 97$$
$$x = 8^{2^2} \ mod \ 97 = 22 \ mod \ 97$$
$$x = 8^{2^3} \ mod \ 97 = 96 \ mod \ 97 = -1 \ mod \ 97$$

Quindi 97 è un probabile primo e la probabilità di errore, dato che si è eseguito una sola volta il test è minore di $1/4$. Per ridurre la probabilità di errore si esegue nuovamente il test scegliendo un diverso valore di y, per esempio $y=3$. Si calcola $x=3^3 \ mod \ 97 = 27 \ mod \ 97$. Anche questa volta si è nel caso di $x \not\equiv 1 \ mod \ N$, si controllano gli i:

$$x = 27^{2^0} \ mod \ 97 = 27 \ mod \ 97$$
$$x = 27^{2^1} \ mod \ 97 = 50 \ mod \ 97$$
$$x = 27^{2^2} \ mod \ 97 = 75 \ mod \ 97$$
$$x = 27^{2^3} \ mod \ 97 = 96 \ mod \ 97 = -1 \ mod \ 97$$

Quindi anche in questo caso 97 è un probabile primo e la probabilità di errore si è ridotta di $(1/4)^2$. La probabilità di errore può essere ancora ridotta...

Il calcolo delle successive potenze è critico. Infatti, se eseguite in modo diretti mediante $(N-1)/2$ successive moltiplicazioni di y per sé stesso, queste operazioni richiedono tempo esponenziale nella dimensione di N. Tuttavia in crittografia si utilizza un meccanismo polinomiale, chiamato **algoritmo delle esponenziazione veloce**, per calcolare l'elevamento a potenza: si immagina di voler calcolare $x=y^z \ mod \ s$ con y, z, s interi dello stesso ordine di grandezza, passi:

1. Si scompone l'esponente z in una somma di potenze di 2, quindi $z=\sum_{i=0}^{t} k_i \cdot 2^i$, dove $k_i \in \{0,1\}$, e $t=\lfloor log_2 z \rfloor$
2. Si calcolano tutte le potenze di $y^{2^i} \ mod \ s = (y^{2^{i-1}})^2 \ mod \ s$ con $1 \leq i \leq t$

3. Si calcola $x = y^z \ mod \ s$ come $x = \Big(\prod_{i=k_i \neq 0} y^{2^i} \ mod \ s \Big) \ mod \ s$

Esempio. Si suppone di voler calcolare $x = 9^{45} \ mod \ 11$

1. Si riscrive l'esponente (partendo da 2^t) ovvero $9^{32+8+4+1} \ mod \ 11$
2. Ora si calcolano le potenze 9^{2^i} fino ad arrivare a 9^{32} e ciascuna potenza la si calcola come il quadrato della precedente:

$$
\begin{aligned}
&9^2 \ mod \ 11 = 4 \\
&9^4 \ = 4^2 \ mod \ 11 = 5 \\
&9^8 \ = 5^2 \ mod \ 11 = 3 \\
&9^{16} = 3^2 \ mod \ 11 = 9 \\
&9^{32} = 9^2 \ mod \ 11 = 7
\end{aligned}
$$

   in generale si fanno t moltiplicazioni, dove $t = \lfloor log_2 z \rfloor$
3. Ora per calcolare 9^{45} basta utilizzare le potenze che servono:

$$9^{45} = ((9^{32} \ mod \ 11)(9^8 \ mod \ 11)(9^4 \ mod \ 11)(9^1 \ mod \ 11)) \ mod \ 11 = (4 \cdot 3 \cdot 5 \cdot 9) \ mod \ 11 = 1$$

Nel passo 2. si svolgono $\Theta(logz)$ moltiplicazioni, mentre nel passo 3. $O(logz)$. Ogni moltiplicazione ha un costo al più quadratico nel numero delle cifre. Quindi l'algoritmo è polinomiale nella dimensione dei dati.

3.3.2 Generazione di numeri primi

Il contributo degli algoritmi randomizzati alla crittografia è determinante quanto meno nella generazioni di numeri binari primi grandissimi. Per generare un numero primo, si genera un numero casuale, seguito da un test di primalità. Se il test fallisce, si ripete fino a quando si trova un numero dichiarato primo (con probabilità di errore minore di $(1/4)^k$. Sono pochi i numeri da testare grazie alla densità: il numero di interi primi e minori di N tendono a $N/log_e N$ per N che tende all'infinito. Quindi per N sufficientemente grande, in un suo intorno di ampiezza $log_e N$ cade mediamente un numero primo. E poiché $log_e N$ è proporzionale alla dimensione n di N, il numero di prove attese è polinomiale in n. Si può allora definire il seguente algoritmo `Primo(n)` per la generazione (probabilità di errore $(1/4)^k$) di un numero primo binario di almeno n bit, dove n è un valore fissato dall'utente

```
Primo(n)
S = sequenza casuale di n-2 bit
N = 1 S 1
while(Test-MR(N, k)==0) N = N + 2
return N
```

L'algoritmo è complessivamente polinomiale $O(n^4)$

3.4 Classe RP

La classe **random polinomiale** è la classe dei problemi decisionali verificabili in tempo polinomiale randomizzato. Dato un problema Π decisionale e x come sua istanza allora si dice che y è un **certificato probabilistico** di x se y è di lunghezza al più polinomiale in $|x|$ e y è estratto perfettamente a caso da un insieme associato a x. Si chiama $A(x, y)$ **algoritmo di verifica** polinomiale che attesta con certezza che x non possiede la proprietà esaminata da Π, ovvero $\Pi(x) = 0$, oppure attesta che x possiede la proprietà esaminata da Π con probabilità maggiore di $1/2$. La relazione tra le principali classi diviene: $P \subseteq RP \subseteq NP$

Chapter 4

Cifrari storici

4.1 Cifrario di Cesare

I cifrari storici sono stati già forzati grazie a tecniche in verità molto semplici. Tuttavia alcune antiche idee sono state riprese nei sofisticati metodi di oggi. La cifratura e decifrazione erano realizzati con carta e penna e i messaggi da cifrare erano frasi di senso compiuto in linguaggio naturale. Quasi tutti i cifrari storici, nella loro progettazione, seguono i **principi di Bacone**:

1. Le funzioni $\mathcal{C}$ e $\mathcal{D}$ devono essere facili da calcolare
2. È impossibile ricavare la $\mathcal{D}$ se la $\mathcal{C}$ non è nota
3. Il crittogramma $c = \mathcal{C}(m)$ deve apparire "innocente"

Il **cifrario di Cesare** è il più antico cifrario di concezione moderna. *Idea di base*: il crittogramma c è ottenuto dal messaggio in chiaro m sostituendo ogni lettera con quella tre posizioni più avanti nell'alfabeto. Le caratteristiche principali di questo cifrario sono che non ha una chiave segreta, la segretezza dipendeva dalla conoscenza del metodo, scoprire il metodo di cifratura significa comprometterne irrimediabilmente l'impiego, quindi il cifrario era destinato all'uso ristretto di un gruppo di conoscenti.

4.1.1 Cifrario di Cesare generalizzato

Per rendere il cifrario di Cesare più sicuro si può, invece di ruotare l'alfabeto di 3 posizioni, lo si può ruotare di una quantità arbitraria k con $1 \leq k \leq 25$. In questo caso k è la chiave segreta del cifrario. **Formulazione matematica**: si indica con $pos(X)$ la posizione nell'alfabeto della lettera X. La cifratura del carattere X corrisponde a quel carattere Y tale che $pos(Y) = (pos(X) + k)\ mod\ 26$. La funzione di decifrazione di Y è analoga, ovvero, $pos(X) = (pos(Y) - k)\ mod\ 26$. Se si conosce la struttura del cifrario, si possono applicare in breve tempo tutte le chiavi possibili (25) a un crittogramma per decifrarlo e scoprire contemporaneamente la chiave segreta. Quindi è un cifrario inutilizzabile a fini crittografici. Gode della proprietà commutativa: data una sequenza di chiavi e di operazioni di cifratura e decifrazione, l'ordine delle operazioni può essere permutato arbitrariamente senza modificare il crittogramma finale. Per aumentare ancora la sicurezza, si può essere tentati di ripetere più volte l'operazione di cifratura ma per come è costruito questo cifrario, se si applica la funzione di cifratura due volte è come fare una singola operazione di cifratura in cui lo shift è la somma dei due k. Non sempre comporre più cifrari aumenta la sicurezza del sistema

4.2 Classificazione dei cifrari storici

Ci sono due grandi famiglie:

- **Cifrari a sostituzione**, sostituiscono ogni lettera del messaggio in chiaro con una o più lettere dell'alfabeto secondo una regola prefissata. I cifrari a sostituzione si dividono a loro volta in:
  - **Sostituzione monoalfabetica**
    alla stessa lettera del messaggio corrisponde sempre una stessa lettera nel crittogramma. Esempio: cifrario di Cesare. Si possono impiegare funzioni di cifratura e decifrazione più complesse dell'addizione e della sottrazione in modulo. La spazio delle chiavi è molto più ampio ma la sicurezza è comunque molto modesta. Come esempio si studia il cifrario affine:

Cifrario affine. *Cifratura*, una lettera in chiaro X viene sostituita con la lettera cifrata Y che occupa nell'alfabeto la posizione $pos(Y) = (a \cdot pos(X) + b)\ mod\ 26$. Se $a = 1$ si ha esattamente il cifrario di cesare generalizzato dove b gioca il ruolo di k. La chiave segreta del cifrario k è a e b

Decifrazione, è data da $pos(X) = a^{-1} \cdot (pos(Y) - b)\ mod\ 26$, con a^{-1} inverso di a modulo 26. L'inverso di un intero a modulo m esiste ed è unico se e solo se $MCD(a, m) = 1$. Questo è un vincolo forte sulla definizione della chiave perché se $MCD(a, 26) \neq 1$ la funzione di cifratura non è iniettiva, e la decifrazione diventa impossibile. Quindi a e 26 devono essere coprimi: i fattori primi di 26 sono 2 e 13 quindi a può assumere qualsiasi valore dispari tra 1 e 25, ad eccezione di 13 (12 valori possibili). b può essere scelto liberamente tra 0 e 25 (26 valori possibili). Le chiavi legittime, ovvero le coppie $\langle a, b \rangle$ sono $12 \cdot 26 = 312$, in realtà 311 perché la coppia $\langle 1, 0 \rangle$ lascia inalterato il messaggio.

Se la segretezza dipende unicamente dalla chiave, il numero delle chiavi deve essere così grande da essere praticamente immune da ogni tentativo di provarle tutte. La chiave segreta deve essere scelta in modo casuale. Si deve dunque aumentare il numero di permutazioni realizzabili con la chiave segreta del cifrario. Per raggiungere questo obiettivo si utilizza un **cifrario completo**, dove si assume che la chiave segreta k sia un'arbitraria permutazione delle lettere dell'alfabeto. In questo modo la lettera in chiaro che occupa la posizione i nell'alfabeto è trasformata nella lettera che occupa la posizione i nella permutazione scelta. Le chiavi possibili sono $26! - 1$ (escludendo la permutazione identica). Il cifrario non è comunque sicuro perché si può forzare, senza ricorrere a un attacco esauriente, sfruttando la struttura logica dei messaggi in chiaro e l'occorrenza statistica delle lettere

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
SDTKBJOHRZCUNYEPXVFWAGQILM
```

testo: BOBSTAIATTENTOAEVE
messaggio cifrato: DEDFWSRSWWBYWESBGB

– **Sostituzione polialfabetica**
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. L'esempio più antico è il cifrario di Augusto

Cifrario di Augusto. I documenti erano scritti in numeri, non in lettere. Augusto li scriveva in greco, poi metteva in corrispondenza la sequenza di lettere del documento con la sequenza di lettere del primo libro dell'Iliade. 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. Per esempio se la lettera in posizione i nel documento è α e la lettera in posizione i nell'Iliade è ϵ allora la distanza nell'alfabeto greco tra α e ϵ è 4, e quindi questo numero è la cifratura di quell'α. Il cifrario di Augusto è un cifrario difficile da forzare se la chiave è lunghissima. Lo svantaggio è quello di dover registrare per iscritto la chiave per poterla ricordare

Un altro esempio importante è il disco cifrante di Alberti.

Cifrario di Alberti. Era basato su due dischi concentrici che mittente e destinatario dovevano possedere uguale (e diverso da quello di altre coppie di utenti). Il disco esterno era costituito da un insieme di caratteri un po' limitato disposti in ordine alfabetico, seguito da alcune cifre numeriche: con esso si formulava il messaggio e le cifre erano utilizzate per variare liberamente la chiave durante la cifratura. Il disco interno conteneva un alfabeto più ricco disposto in ordine arbitrario proprio di ogni coppia di dischi e non conteneva cifre: con questo si costruiva il crittogramma. Il continuo cambio di chiave rende inutili gli attacchi basati sulla frequenza dei caratteri. La **macchina Enigma** è un'estensione elettromeccanica del cifrario di Alberti.

```
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 O N F I D A 2 R T I D I E V E
c = U N U J R K S Q
```

qui la chiave diventa A–Q

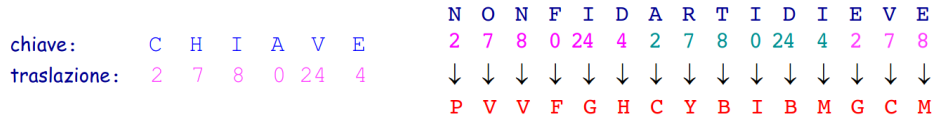
```
A B C D E F G H I L M N O P Q R S T U V Z 1 2 3 4 5
Q I L M S D T K B J O H R Z C U N Y E P X V F W A G
```

Chiave: A–Q
Messaggio: NON FIDARTI DI EVE

```
m = N O N 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
```

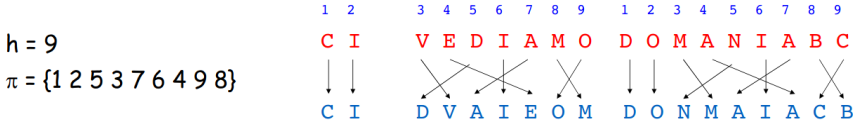
qui la chiave diventa A–Q

Cifrario di Vigenère. È meno sofisticato di quello di alberti ma è più pratico: la chiave è corta e ripetuta ciclicamente. Ogni lettera della chiave indica una traslazione della corrispondente lettera del testo. Una stessa lettera in chiaro viene cifrato in modi diversi, a seconda della lettera della chiave a cui è allineata. Di fatto è una sequenza di cifrari di Cesare con shift diversi. La sicurezza del metodo è influenzata dalla lunghezza della chiave: se la chiave contiene h caratteri, le apparizioni della stessa lettera distanti un multiplo di h nel messaggio si sovrappongono alla stessa lettera della chiave, quindi sono trasformate nella stessa lettera cifrata. Se si estende il metodo di Vigenère impiegando una chiave lunga come il testo, casuale e non riutilizzabile, il cifrario diviene inattaccabile: **one-time pad**



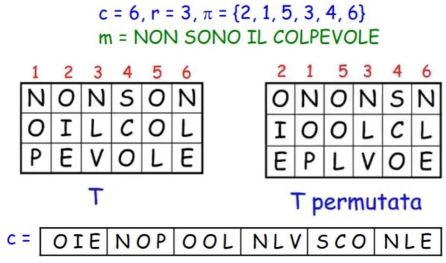
- **Cifrari a trasposizione**, permutano le lettere del messaggio in chiaro secondo una regola prefissata. *Idea di base*: eliminare qualsiasi struttura linguistica presente nel crittogramma permutando le lettere del messaggio in chiaro e inserendone eventualmente altre che vengono ignorate nella decifrazione. Il più semplice cifrario a trasposizione è quello a permutazione semplice

Cifrario a permutazione semplice. La chiave è un intero h e una permutazione π degli interi $\{1, 2..., h\}$. Si suddivide il messaggio m in blocchi di h lettere e si permutano le lettere di ciascun blocco secondo π. Se la lunghezza di m non è divisibile per h si aggiungono alla fine delle lettere qualsiasi che partecipano alla trasposizione ma sono ignorate dal destinatario perché la decifrazione le riporta alla fine del messaggio. Le chiavi sono $h! - 1$ dove h non è fissato a priori. Tanto maggiore è h, tanto più difficile è impostare un attacco esauriente. Al crescere di h cresce anche la difficoltà di ricordare la chiave π



Un interessante generalizzazione del cifrario precedente è il cifrario a permutazione di colonne

Cifrario a permutazione di colonne. La chiave segreta $k = \langle c, r, \pi \rangle$ consiste in due interi c e r che denotano rispettivamente il numero di colonne e di righe di una tabella di lavoro T, e di una permutazione π degli interi $\{1, 2, ..., c\}$. Il messaggio in chiaro m viene decomposto in blocchi $m_1, m_2, ...$ di $c \times r$ caratteri ciascuno. Nella cifratura di un generico blocco m, i caratteri sono distribuiti tra le celle di T in modo regolare, scrivendoli per righe dall'alto verso il basso. Le colonne di T vengono permutate secondo π e lette dall'alto verso il basso e da sinistra verso destra. Per cifrare il blocco successivo, si azzera T e si ripete il procedimento. Le chiavi sono teoricamente esponenziali nella lunghezza del messaggio non essendoci vincoli su r e c



Cifrario a griglia. La chiave segreta è una griglia quadrata $q \times q$, con q pari, in cui $s = q^2/4$ celle sono trasparenti e le altre opache. Si scrivono i primi s caratteri del messaggio, nelle posizioni corrispondenti alle celle trasparenti. Dopo la griglia viene rotata tre volte di 90° in senso orario, e si ripete per ogni rotazione l'operazione di scrittura di tre successivi gruppi di s caratteri. Vi sono $2^{q^2/2}$ griglie (cioè chiavi segrete) possibili

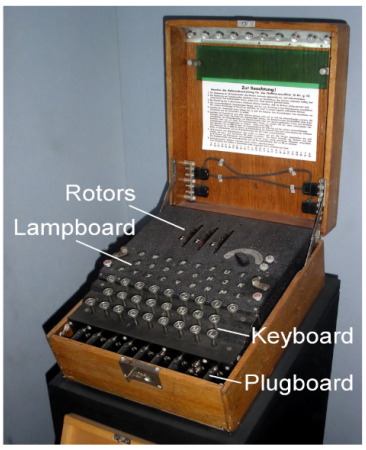
4.3 Crittoanalisi statica

La sicurezza di un cifrario è legata alla dimensione dello spazio delle chiavi che deve essere sufficientemente ampio da impedire attacchi brute force. Ciò non esclude però la possibilità che il cifrario venga attaccato, non da un punto di vista algoritmico, ma statistico (analisi delle frequenze). 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. Spesso la vulnerabilità non è nell'algoritmo, ma nelle sue applicazioni: chiavi usare male (troppo corte, prevedibili, generate male, ...) o si conosce il formato del messaggio. Si ipotizza che il crittoanalista conosca il metodo impiegato per la cifratura/decifrazione, che il messaggio sia scritto in un linguaggio naturale noto al crittoanalista e che il messaggio sia sufficientemente lungo per poter rilevare alcuni dati statistici sui caratteri che compongono il crittogramma. L'attacco si basa sulla frequenza con cui appaiono in media le varie lettere dell'alfabeto in ogni lingua. Dati simili sono noti per le frequenze di: **digrammi** (gruppi di due lettere consecutive), **trigrammi** (gruppi di tre lettere consecutive) e così via **q-grammi**. Ora se un crittogramma è stato generato per sostituzione monoalfabetica la frequenza con cui vi appare una lettera y, corrispondente alla x del messaggio, sarà approssimativamente uguale alla frequenza della x in italiano. Confrontando le frequenze delle lettere come appaiono nel crittogramma con quelle dell'italiano, e provando alcune permutazioni tra lettere di frequenze assai prossime, si ottengono diverse ipotesi di prima decifrazione in genere non lontane dalla realtà.

- **Decifrazione di cifrari a sostituzione monoalfabetica**
  - **Cifrario di Cesare**, è sufficiente scoprire la corrispondenza di una sola coppia $\langle y, x \rangle$ per svelare l'intero messaggio
  - **Cifrari affini**, è sufficiente individuare due coppie di lettere corrispondenti con cui impostare un sistema di due equazioni nelle due incognite a e b che formano la chiave segreta. La risoluzione del sistema è sempre possibile perché a è invertibile
  - **Cifrari completi**, si associano le lettere in base alle frequenze
- **Decifrazione di cifrari a sostituzione polialfabetica**, la decifrazione è più difficile: l'istogramma delle frequenze calcolato sul testo cifrato risulta appiattito
  - **Cifrario di Vigenère**, ogni lettera y del crittogramma dipende da una coppia di lettere $\langle x, k \rangle$ provenienti dal messaggio e dalla chiave. La debolezza è che la chiave unica è ripetuta più volte. Se la chiave contiene h caratteri il crittogramma può essere decomposto in h sottosequenze ciascuna delle quali è stata ottenuta per sostituzione monoalfabetica: il problema è dunque quello di scoprire il valore di h per scomporre il crittogramma e continuare poi la decifrazione con il metodo monoalfabetico. *Se si conosce* h: si scompone il crittogramma in h sottomessaggi: si prende il carattere di posizione 1 e tutti i caratteri a distanza h ovvero $1 + h$, $1 + 2h$, ... e così via per il carattere in posizione 2 e gli altri. In ciascuno di tali sottomessaggi tutte le lettere sono allineate con la stessa lettera della chiave, quindi la cifratura di ciascun sottomessaggio è di fatto una cifratura in stile cifrario di Cesare generalizzato. I cifrari polialfabetici non sono dunque tanto più robusti dei monoalfabetici se le chiavi non sono molto lunghe. *Se non si conosce* h: il messaggio contiene quasi sicuramente gruppi di lettere adiacenti ripetuti più volte. Apparizioni della stessa sottosequenza allineate con la stessa porzione della chiave sono trasformate nel crittogrammma in sottosequenze identiche. Si cercano nel crittogramma coppie di posizioni p_1, p_2 in cui iniziano sottosequenze identiche. La distanza $d = p_2 - p_1$ è probabilmente uguale alla lunghezza h della chiave, o a un suo multiplo
  - **Cifrario di Alberti**, immune da questi attacchi se la chiavi se la chiave viene cambiata spesso evitando pattern ripetitivi. Mantenere a lungo una chiave mette a rischio il cifrario perché in quel tratto la sostituzione è monoalfabetica
- **Decifrazione cifrari a trasposizione**, le lettere nel crittogramma sono le stesse del messaggio in chiaro, quindi non ha senso condurre un attacco statistico basato sulle frequenze. Un aiuto viene dallo studio dei q-grammi. Se si conosce la lunghezza h della chiave: si divide il crittogramma in porzioni di lunghezza h; in ciascuna si cercano i gruppi di q lettere che formano i q-grammi più diffusi nel linguaggio; se un gruppo deriva effettivamente da un q-gramma, si scopre parte della permutazione

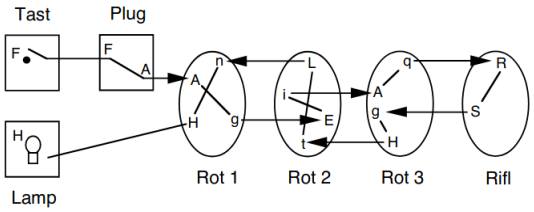
In conclusione la rilevazione delle frequenze delle singole lettere del crittogramma è un potente indizio per discernere tra i vari tipi del cifrario. L'analisi delle frequenze può aiutare a capire il metodo di cifratura. Per esempio nei *cifrari a trasposizione* l'istogramma delle frequenze coincide approssimativamente con quello del proprio linguaggio; nei *cifrari a sostituzione monoalfabetica* i due istogrammi coincidono a meno di una permutazione delle lettere e nei *cifrari a sostituzione polialfabetica* l'istogramma del crittogramma è assai più appiattito di quello del linguaggio

4.4 Macchina Enigma



I numerosissimi sistemi crittografici usati dall'uomo hanno subito un epocale rinnovamento con l'apparizione dei calcolatori elettronici. Il sistema **Enigma** sviluppato in Germania a partire dal 1918, per applicazioni commerciali, ha avuto un ruolo fondamentale nella seconda guerra mondiale. Non trasmetteva i messaggi ma era un dispositivo per costruire il crittogramma. 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 e di ventisei lampadine corrispondenti alle lettere dell'alfabeto tedesco. 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 riversibile, il ricevente batteva sulla tastiera il crittogramma per ricostruire il messaggio sulle lampadine. Il cuore di Enigma era costituito da tre rotori (dischi) di gomma rigida montati sullo stesso asse ma liberi di rotare in modo indipendente, e da un riflettore (disco fisso). 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 commutazione che ne incrementava la sicurezza. Le due facce di ogni rotore dette rispettivamente **pad** e **pin** erano dotate di 26 contatti elettrici disposti in cerchio corrispondenti ai caratteri dell'alfabeto. Il rotore conteneva tra pad e pin una permutazione fissa delle lettere. Una caratteristica fondamentale era che i rotori non mantenevano la stessa posizione reciproca durante la cifratura, ma per ogni lettera battuta dall'operatore il primo rotore avanzava di un passo; dopo 26 passi il rotore era tornato nella posizione iniziale e avanzava di un passo il secondo rotore; quando anche questo aveva fatto una rotazione completa avanzava di un passo il terzo rotore. La corrispondenza tra caratteri cambiava ad ogni passo, ovvero, la chiave cambiava a ogni passo.

Esempio. Si immagina di digitare la lettere F, il pannello di commutazione scambia la lettera con un'altra per esempio la A. Dopodiché la A farà contatto con la A del primo rotore. Ogni rotore contiene un cablaggio interno che mappa ogni lettera che occorre su una faccia del rotore con una lettera diversa sulla faccia posteriore del rotore. Quindi si suppone che internamente esista un cavo che collega il contatto elettrico corrispondente alla A del pad con il contatto elettrico corrispondente alla g del pin. Poi il contatto di g toccherà il corrispondente contatto sul pad del secondo rotore, che si suppone sia la lettera E. Quest'ultima viene collegata internamente alla i, che a sua volta si collega con un'altra A del terzo rotore. La A è cablata con la lettera q che è collegata alla lettera R del riflettore. Il riflettore è come se fosse un rotore ma è fisso con una singola facciata. Si immagina che la R sia collegata con la S e dopodiché si torna indietro, ovvero si effettua il percorso dal riflettore fino al primo rotore che come ultima lettera ha H e quindi si accede la lampadina H



Se i rotori fossero fermi si avrebbe in una sola chiave monoalfabetica la cui sicurezza, come si è visto, è praticamente nulla. Poiché però i rotori si muovono uno rispetto all'altro la chiave cambia a ogni passo: 26 permutazioni si ottengono con le rotazioni del primo rotore rispetto al secondo; 26 tra il secondo e il terzo; 26 tra il terzo e il riflettore; per un totale di $26 \times 26 \times 26 = 17576$ chiavi diverse in un assetto polialfabetico. La debolezza sta nel fatto che i rotori sono oggetti fisici immutabili, le 26^3 permutazioni sono sempre le stesse, applicate nello stesso ordine. Un altro problema è che le chiavi erano note a tutti i proprietari di Enigma. Alberti aveva previsto, per ogni coppia di utenti, una coppia di dischi diversa da tutte le altre.

4.4.1 Modifiche

Le debolezze di enigma furono risolte con vari meccanismi che resero la rottura del cifrario molto ardua. Anzitutto la macchina fu costruita in modo che i tre rotori potessero essere permutati tra loro quindi il numero di permutazioni diventa $3! \times 26^3 > 10^5$, e la serie di chiavi dipende dalla permutazione scelta. Inoltre fu aggiunto tra tastiera e primo

rotore un *plugboard* (pannello di commutazione) che consentiva di scambiare tra loro i caratteri di 6 coppie scelte arbitrariamente in ogni trasmissione. In tal modo, considerando che ogni cablaggio è descritto da una sequenza di 12 caratteri (le 6 coppie da scambiare), si potevano realizzare $\binom{26}{12}$ combinazione possibili. Ogni gruppo di 12 caratteri si può presentare in $12!$ permutazioni diverse, ma non tutte producono effetti diversi. Per esempio:

$$\begin{array}{cccccc} AB & CD & EF & GH & IJ & KL \\ CD & AB & EF & GH & IJ & KL \end{array}$$

producono lo stesso effetto, e con queste anche tutte le $6!$ *permutazioni delle* 6 *coppie*

Infine si devono considerare i possibili scambi tra gli elementi delle coppie, che producono lo stesso effetto:

$$\begin{array}{cccccc} AB & CD & EF & GH & IJ & KL \\ BA & CD & EF & HG & IJ & KL \end{array}$$

si deve dividere per un ulteriore fattore $2^6 = 64$

In complesso l'introduzione del plugboard moltiplica il numero di chiavi dei rotori per un fattore $\binom{26}{12}\frac{12!}{6!\cdot 64} > 10^{11}$ per un totale di più di 10^{16} chiavi. 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; e aumentarono a dieci le coppie di caratteri scambiabili nel plugboard. Come conseguenza di tutti questi modi di funzionamento 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 elenco segreto 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 5

Cifrari perfetti

5.1 Definizione

Un **cifrario** è **perfetto** se la sua sicurezza è garantita qualunque sia l'informazione catturata dal crittoanalista. Per formalizzare si introducono due spazi:

MSG, spazio dei messaggi

$CRITTO$, spazio dei crittogrammi

e due variabili aleatorie:

M, descrive il comportamento del mittente, e assume valori in MSG

C, descrive la comunicazione sul canale, e assume valori in $CRITTO$

Per ogni $m \in MSG$ e per ogni $c \in CRITTO$ si definisce:

$P(M = m)$, probabilità che il mittente voglia inviare il messaggio m

$P(M = m \mid C = c)$, probabilità a posteriori che il messaggio inviato sia m posto che sul canale transita il crittogramma c

Per accertare l'assoluta sicurezza del sistema crittografico, si studia la situazione di massimo pessimismo in cui il crittoanalista che ha intercettato c sia in possesso di tutta l'informazione possibile sul sistema tranne la chiave segreta. Si riporta ora la **definizione di Shannon**: un cifrario è perfetto se per ogni $m \in MSG$ e per ogni $c \in CRITTO$ vale la relazione $P(M = m \mid C = c) = P(M = m)$. Una immediata conseguenza della definizione è che impiegando un cifrario perfetto la conoscenza complessiva del crittoanalista non cambia dopo che egli ha osservato un crittogramma arbitrario c transitare sul canale

Esempio 1. Si immagina ci sia un certo messaggio $\overline{m} \in MSG$ che ha questa caratteristica $P(M = \overline{m}) = p > 0$, con $0 < p < 1$. Ora si suppone che esista un crittogramma $\overline{c}$ tale che $P(M = \overline{m} \mid C = \overline{c}) = 1$. Il cifrario non è perfetto perché le due probabilità sono diverse (la prima è minore di 1, mentre la seconda è uguale a 1), in questo esempio il crittoanalista deduce che il messaggio spedito è proprio $\overline{m}$

Esempio 2. Si immagina ancora che esista un certo messaggio $\overline{m} \in MSG$ con la caratteristica $P(M = \overline{m}) = p > 0$, con $0 < p < 1$. Adesso si suppone che esiste il crittogramma $\overline{c}$ con $P(M = \overline{m} \mid C = \overline{c}) = 0$. Anche in questo caso il crittoanalista raffina la sua conoscenza perché deduce che il messaggio spedito non è m, mentre prima sapeva che sarebbe potuto transitare con probabilità p.

In tutti i casi intermedi il crittoanalista raffina la sua conoscenza sul possibile messaggio spedito: tale conoscenza rimane inalterata solo se $P(M = m \mid C = c) = P(M = m)$. Dalla definizione di Shannon discende il **teorema di Shannon** che sottolinea la non praticità dei cifrari perfetti: in un cifrario perfetto il numero delle chiavi deve essere maggiore o uguale al numero dei messaggi possibili

Dimostrazione. Sia N_m il numero di messaggi possibili, cioè tali che $P(M = m) > 0$, e sia N_k il numero di chiavi. Si pone per assurdo che sia $N_m > N_k$. A un crittogramma c, con $P(C = c) > 0$ corrispondono $s \leq N_k$ messaggi ottenuti decrittando c con tutte le chiavi, dove s è il numero di messaggi che possono corrispondere al crittogramma c. Poiché $N_m > N_k \geq s$, esiste almeno un messaggio m con $P(M = m) > 0$ non ottenibile da c. Questo implica $P(M = m \mid C = c) = 0 \neq P(M = m)$ ovvero per $N_m > N_k$ il cifrario non è perfetto

5.2 Cifrario One-Time Pad

One-Time Pad è un cifrario perfetto molto semplice e veloce. Lo spazio dei messaggi, dei crittogrammi e delle chiavi sono sequenze di n bit. Dato un messaggio m di n bit e una chiave k, la cifratura produrrà un crittogramma c anch'esso di n bit. Le funzioni di cifratura e di decifrazione sono basate sullo XOR:

Generazione della chiave segreta, si costruisce una sequenza $k = k_1, k_2, ...$ di bit nota a mittente e a destinatario, avente lunghezza maggiore o uguale a quella del messaggio da scambiare. Ogni bit di k_i di k è scelto perfettamente a caso tra 0 e 1

Cifratura, se il messaggio da trasmettere è la sequenza di n bit $m = m_1 m_2, ..., m_n$, il crittogramma $c = c_1 c_2, ..., c_n$ si genera bit a bit ponendo $c_i = m_i \oplus k_i$ per $i = 1, 2, ..., n$

Decifrazione, presi gli n bit iniziali $k_1 k_2, ..., k_n$ della chiave segreta e il crittogramma c, il messaggio m è ricostruito bit a bit ponendo $m_i = c_i \oplus k_i$, per $i = 1, 2, ..., n$

Se la chiave non è scelta perfettamente a caso, ma, possiede qualche regolarità, lo XOR farebbe filtrare tale proprietà sul crittogramma. Esempio:

Cifratura:

m	=	0	1	0	1	0	1	0	1	
k	=	0	1	0	0	1	1	1	0	...
c	=	0	0	0	1	1	0	1	1	

Decifrazione:

c	=	0	0	0	1	1	0	1	1	
k	=	0	1	0	0	1	1	1	0	...
m	=	0	1	0	1	0	1	0	1	

È fondamentale non riutilizzare la chiave: si ha un messaggio m_1 che viene inviato usando una chiave k, quindi $c_1 = m_1 \oplus k$. Poi il mittente, riutilizzando la stessa chiave invia secondo messaggio $c_2 = m_2 \oplus k$. Il crittoanalista, sapendo che il Alice e Bob riusano la stessa chiave, può calcolare $c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = (m_1 \oplus m_2) \oplus (k \oplus k) = (m_1 \oplus m_2) \oplus 0 = m_1 \oplus m_2$. Si dimostra che il One-Time Pad è perfetto utilizzando le seguenti ipotesi:

1. Tutti i messaggi hanno lunghezza n. Si effettua il *padding* se la lunghezza del messaggio è minore di n (ovvero se il messaggio è breve lo si allunga fino a n) o si fa una cifratura a blocchi lunghi n se il messaggio è più lungo di n
2. Tutte le sequenze di n bit sono messaggi possibili. Si assegna una probabilità molto bassa ma maggiore di 0 alle sequenze binarie prive di significato
3. La chiave è scelta perfettamente a caso per ogni messaggio

Sotto le ipotesi 1, 2 e 3, One-Time Pad è un cifrario perfetto e impiega un numero minimo di chiavi

Dimostrazione 1. *Minimalità,* segue immediatamente dal fatto che $N_m = N_k = 2^n$

Dimostrazione 2. *Cifrario perfetto*: si deve provare $\forall m \in MSG$ e $\forall c \in CRITTO$ allora $P(M = m \mid C = c) = P(M = m)$. Applicando la definizione di probabilità condizionale si può riscrivere il termine sinistro della formula come: $P(M = M \mid C = c) = \frac{P(M=m, C=c)}{P(C=c)}$. L'evento $\{M = m, C = c\}$ descrive la situazione in cui il mittente ha generato il messaggio m e l'ha cifrato come crittogramma c. Per la definizione di XOR fissato il messaggio, chiavi diverse danno origine a crittogrammi diversi e esiste un'unica chiave k che porta un messaggio m fissato in un certo crittogramma c. Quindi la probabilità che il crittogramma sia c, $P(C = c)$, è uguale alla probabilità di scegliere a caso l'unica chiave che porta m in c e siccome la chiave viene scelta a caso, ed è lunga n bit, la probabilità di ottenere quel crittogramma c è $(1/2)^n$: $P(C = c) = (1/2)^n$. Dunque la probabilità di ottenere il crittogramma c è sempre $(1/2)^n$ ed è indipendente da m, cioè gli eventi $\{M = m\}$ e $\{C = c\}$ sono indipendenti tra loro (in eventi indipendenti si moltiplica) e si ha: $P(M = M \mid C = c) = \frac{P(M=m, C=c)}{P(C=c)} = \frac{P(M=m)\cancel{P(C=c)}}{\cancel{P(C=c)}} = P(M = m)$ cioè il cifrario One-Time Pad è perfetto.

Si studia quanto si riesce a mantenere delle proprietà del cifrario e se si può "risparmiare" qualche bit della chiave k: si prova a rimuovere l'ipotesi 2, e considerare solo messaggi significativi. Nella lingua inglese i messaggi significativi sono circa α^n con $\alpha = 1.1$. Il numero delle chiavi $N_k \geq N_m = \alpha^n < 2^n$. Si immagina di poter descrivere N_k con t bit, con t tale che $2^t \geq \alpha^n$, trovando $t \geq nlog_2\alpha \simeq 0.12n$: il numero di bit della chiave passa così da n a $0.12n$. Per confondere il crittoanalista è opportuno fare in modo che molte coppie (m, k) diverse producano lo stesso crittogramma, in questo modo se il crittoanalista effettua il metodo brute force alla ricerca di messaggi significativi ne trova tanti e non conosce qual è l'm effettivo. Per garantire che tante coppie (m, k) finiscono sullo stesso crittogramma si impone che il numero delle coppie (m, k) sia molto più ampio del numero dei crittogrammi, cioè $\alpha^n 2^t >> 2^n = nlog_2\alpha + t >> n$ quindi $t >> n - nlog_2\alpha = n(1 - 0.12) \Rightarrow t >> 0.88n$. Dunque la lunghezza della chiave rimane necessariamente assai prossima a n anche sotto queste ipotesi restrittive. Lo svantaggio principale del cifrario one-time pad è che la chiave deve essere perfettamente casuale e deve essere utilizzata una sola volta

Chapter 6

Cifrari per le comunicazioni di massa

6.1 Caratteristiche cifrari simmetrici

Il One-Time Pad non è un cifrario per le comunicazioni di massa perché richiede una nuova chiave segreta per ogni messaggio che deve essere perfettamente casuale e lunga quanto il messaggio da scambiare. Il One-Time Pad protegge il messaggio con certezza assoluta. I cifrari per le comunicazioni di massa offrono **sicurezza computazionale** cioè l'informazione è protetta se il crittoanalista ha accesso a risorse computazionali limitate (polinomiali) e se $P \neq NP$. I cifrari simmetrici sono progettati in modo da rispettare i **principi di Shannon**:

- **Diffusione**, il testo in chiaro si deve distribuire su tutto il crittogramma, cioè, ogni carattere del crittogramma deve dipendere da tutti i caratteri del blocco di messaggio
- **Confusione**, messaggio e chiave sono combinati tra loro in modo complesso per non permettere al crittoanalista di separare due sequenze tramite l'analisi statica del crittogramma. La chiave deve essere ben distribuita sul testo cifrato e ogni bit del crittogramma deve dipendere da tutti i bit della chiave

6.2 DES

Nel 1972 il *NBS*, ufficio americano per la definizione degli standard, iniziò un programma per proteggere le comunicazioni non classificate, cioè in genere le comunicazioni commerciali. L'NBS pubblico un bando che richiedeva:

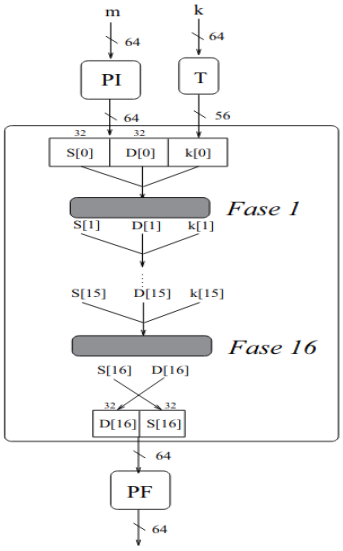
- la sicurezza dell'algoritmo risedesse nella segretezza della chiave e non nel processo di cifratura e decifrazione
- l'algoritmo potesse essere realizzato efficientemente in hardware

Il bando venne accolto dalla *IBM* che propose un sistema, il **DES**, derivato da un sistema chiamato *Lucifer*. La *NSA*, ente che possedeva tutto lo scibile sulle comunicazioni segrete, propose delle variazioni tra cui la riduzione della lunghezza della chiave da 128 a 56 bit e la modifica delle funzioni contenute nella S-box (cuore del cifrario). Dopo una serie di sospetti reciproci, l'IBM accettò le modifiche. A distanza di anni si può affermare che entrambe le parti sono state oneste e i sospetti si sono rilevati infondati. Nel 1977 il DES fu accettato ed è diventato ufficialmente il nuovo standard per la protezione delle comunicazioni non classificate. Ogni 5 anni il DES veniva sottoposto a revisione fino a quando nel 1999 ne fu sconsigliato l'uso, ammettendone però un uso generale nella versione estesa **3DS**. Nel 2005 anche questa versione fu tolta dagli standard. Nel frattempo era già partito il programma per selezionare il nuovo standard: **AES** (2001).

6.2.1 Struttura del DES

Il DES è un cifrario simmetrico di uso generale che presenta la seguente struttura:

- messaggio suddiviso in blocchi da 64 bit ciascuno dei quali viene cifrato e decifrato indipendentemente dagli altri.
- cifratura e decifrazione procedono attraverso 16 fasi (o *round*) successive in cui si ripetono le stesse operazioni
- la chiave è composta di 64 bit di cui 56 sono scelti arbitrariamente e 8 di parità
- dalla chiave k vengono create r sottochiavi impiegate una per ogni fase



Trasformazione Centrale

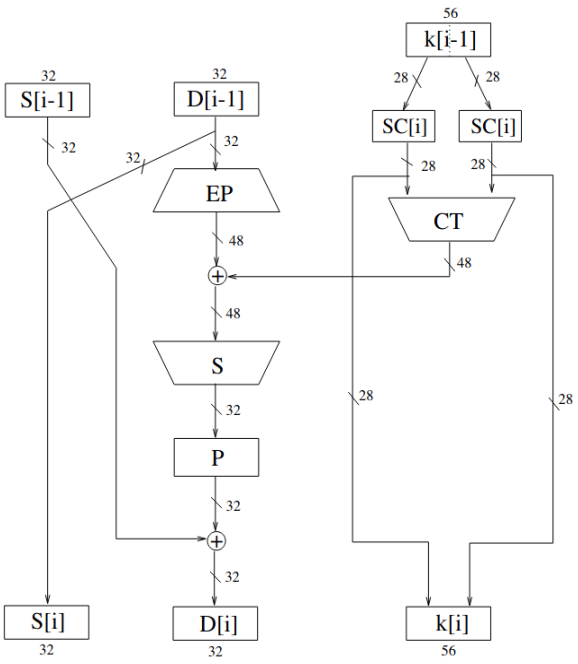
- m: blocco del messaggio
- c: corrispondente blocco del crittogramma
- k: chiave segreta con i bit di parità
- PI: permutazione iniziale del messaggio in chiaro
- PF: permutazione finale inversa di PI
- T: depura e permuta la chiave
- $S[i]$: blocco sinistro
- $D[i]$: blocco destro

In prima approssimazione le operazioni sono schematizzate come segue:

- **Permutazioni iniziali**, PI permuta i 64 bit del messaggio in chiaro. T depura la chiave dai bit di parità e permuta gli altri 56 bit per generare la prima sottochiave $k[0]$
- **Trasformazione centrale**, la sequenza permutata da PI viene decomposta in un blocco sinistro $S[0]$ e un blocco destro $D[0]$ di 32 bit ciascuno. Su questi due blocchi si eseguono, nelle successive fasi del DES sedici trasformazioni strutturalmente uguali, ciascuna con una diversa sottochiave $k[i]$ di 56 bit derivata dalla chiave iniziale k. La i-esima fase riceve in ingresso tre blocchi $S[i-1], D[i-1], k[i-1]$ rispettivamente di 32, 32, 56 bit, calcola $S[i] = D[i-1]$ e $D[i] = S[i-1] \oplus f(D[i-1], k[i-1])$ con f funzione non lineare e produce in uscita tre nuovi blocchi $S[i], D[i], k[i]$ che costituiscono l'ingresso alla fase successiva. 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 da cui si costruirà il crittogramma finale.
- **Permutazione finale**, PF genera la permutazione inversa di PI rimescolando ancora i bit del crittogramma

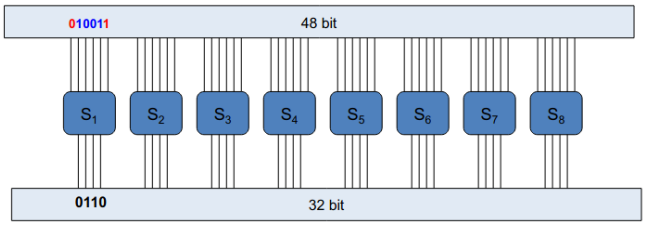
Ora si studiano in dettaglio la fase i-esima del DES:

- **Shift ciclico $SC[i]$**, la sottochiave $k[i-1]$ di 56 bit, ricevuta dalla fase precedente viene suddivisa in 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: $SC[i] = 1$ per $i = 1, 2, 9, 16$, $SC[i] = 2$ altrimenti. Le due parti così traslate vengono concatenate in un unico blocco di 56 bit che costruisce la sottochiave $k[i]$ per la fase successiva
- **Compressione e trasposizione CT**, 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 combinazione delle funzioni $SC[i]$ e CT garantisce che in ogni fase venga estratto dalla sottochiave un diverso sottoinsieme di bit per la cifratura.
- **Espansione e permutazione EP**, il blocco $D[i-1]$ do 32 bit nella fase precedente viene espanso a 48 bit duplicando 16 bit in ingresso e spostandone altri per ottenere un blocco della stessa dimensione di quello estratto dalla sottochiave e per eseguire lo XOR tra i due.
- **S-box S**, è l'unica parte non lineare ed è cruciale per il cifrario. Effettua una trasformazione dei bit e li riduce da 48 a 32 bit
- **Permutazione P**, permutazione di 32 bit che genera il blocco finale $D[i]$



6.2.2 S-box

La S-box è una funzione che riceve 48 bit in input e ne restituisce 32 in output. La S-box è composta da 8 sottofunzioni $S_1, S_2, ..., S_8$ dove ciascuna prende in ingresso 6 bit e ne restituisce 4 in output. Si dividono i bit di input in due sottoinsiemi che definiscono due numeri x, y con $0 \leq x \leq 3$ e $0 \leq y \leq 15$ utilizzati per accedere alla cella di riga x e colonna y in una tabella che definisce la sottofunzione S_i. Per esempio 010011 ha agli estremi i bit 01 (1) e centralmente i bit 1001 (9) che producono come uscita il valore 0110 (06). Non esiste una funzione matematica che calcola cos'è la S-box, ma vi è questa forma tabellare. Questo fatto è stato criticato perché non sapendo matematicamente cosa si sta calcolando è difficile capire se vi è qualche meccanismo segreto



S_1

x \ y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
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

6.2.3 Sicurezza e attacchi

Un cifrario ha una sicurezza di b bit se il costo del miglior attacco è di ordine $O(2^b)$ operazioni di decifrazione, ovvero, richiede di esplorare uno spazio delle chiavi di cardinalità 2^b. I bit di sicurezza del DES sono 56 quindi per forza bruta provando tutte le 2^{56} chiavi possibili il DES è vulnerabile. Alcune osservazioni sulla struttura del cifrario permettono però di ridurre leggermente lo spazio da esplorare: si sfrutta il fatto che $\mathcal{C}_{DES}(m, k) = c$ implica $\mathcal{C}_{DES}(m', k') = c'$ dove m', k' e c' indicano la complementazione bit a bit. Quindi provata una chiave non è necessario ripetere l'operazione sul suo complemento e lo spazio si riduce a 2^{55} chiavi, la metà di tutte quelle possibili.

Dimostrazione. Si effettua un attacco di tipo chosen plain text: il crittoanalista si procura alcune coppie messaggio/crittogramma del tipo (m, c_1) e (m', c_2). Si parte da una di tali coppie e, scelta una chiave k, si calcola $\mathcal{C}_{DES}(m, k)$. Se $\mathcal{C}_{DES}(m, k) = c_1$ probabilmente k è la chiave segreta (non vi è la certezza di questo fatto perché più chiavi potrebbero trasformare m in c_1). Si prova allora k su altre coppie messaggio/crittogramma: se il successo si ripete per alcune volte consecutive si ha la certezza di aver trovato la chiave. Se invece $\mathcal{C}_{DES}(m, k) = c'$, probabilmente k' è la chiave segreta perché genererebbe correttamente la coppia (m', c_2): come nel caso precedente si ripete la prova su altre coppie. Infine i due casi precedenti falliscono quindi né k né k' è la chiave segreta. k e k' si controllano simultaneamente

Nel 1990 è stata diffusa una tecnica di crittoanalisi chiamata **crittoanalisi differenziale**, è di tipo chosen plain text e impiega un insieme di 2^{47} messaggi in chiaro che il crittoanalista ha scelto e di cui è riuscito a procurarsi i relativi crittogrammi. Scegliendo coppie di messaggi con particolari differenze si controlla come rientrano queste differenze nei relativi crittogrammi. Questo permette di assegnare probabilità diverse a chiavi diverse. Procedendo in questo modo, la chiave cercata emerge sulle altre come quella avente probabilità massima. L'uso della crittoanalisi differenziali ha tra l'altro permesso di dimostrare che sedici fasi sono indispensabili se si vuole garantire al DES ragionevole sicurezza, perché questa tecnica permette di attaccare facilmente il cifrario se il numero di fasi è inferiori a sedici. Nel 1993 è stato pubblicato un'altra tecnica di tipo **plain text** (non è scelto il testo in chiaro) che si chiama **crittoanalisi lineare**, permette di inferire alcuni bit della chiave segreta mediante un'approssimazione lineare della funzione di cifratura, mentre i restanti bit sono determinati attraverso un attacco esauriente. Il numero di coppie messaggio/crittogramma da esaminare si riduce a 2^{43}. Il costo è inferiore al forza bruta per cui il DES è vulnerabile alla crittoanalisi lineare. Il colpo finale al DES è stato sferrato nel 2008 con la macchina *RIVYERA*: un calcolatore molto costoso, costruito appositamente per rompere il cifrario in meno di 24 ore

6.2.4 Alternative al DES

Le variazioni del DES hanno lo scopo di preservare la semplicità del cifrario ma rendendolo più robusto. Una possibile idea è quella di usare sottochiavi di fase tutte diverse: le sottochiavi di fase sono 16, ciascuna utilizza 48 bit quindi si scelgono $16 \cdot 48 = 768$ bit diversi, il che rende impraticabile un attacco esauriente sulle chiavi stesse. Tra questi 768 bit non sono tutti bit di sicurezza perché con la crittoanalisi differenziale lo spazio delle chiavi da esplorare ha una ampiezza di 2^{61}. Più rilevante è la **cifratura multipla** che prevede la concatenazione di più copie del DES che utilizzano chiavi diverse. Si costruisce così un cifrario composto il che, come si è già studiato, non garantisce in linea di principio un aumento della sicurezza. Nel caso del DES però, la concatenazione di più copie del cifrario non

è equivalente a un'applicazione singola del cifrario stesso. Matematicamente ciò si esprime con l'affermazione che, date due arbitrarie chiavi k_1, k_2 si ha $\mathcal{C}_{DES}(\mathcal{C}_{DES}(m, k_1), k_2) \neq \mathcal{C}_{DES}(m, k_3)$ per qualsiasi messaggio m e qualsiasi chiave k_3. Due chiavi di 56 bit permettono una chiave di 112 bit ma che in realtà solo 57 sono di sicurezza. Questi 57 bit vengono fuori da un attacco che si chiama **meet in the middle** che funziona in questo modo: il crittogramma, come si è visto, viene cifrato due volte $\mathcal{C}_{DES}(\mathcal{C}_{DES}(m, k_1), k_2)$ ma se è vera questa relazione è vera anche $\mathcal{D}_{DES}(c, k) = \mathcal{D}_{DES}(m, k_1)$. Il crittoanalista parte alla ricerca di una coppia di chiavi k_1, k_2 che ha questa relazione, quindi si procura una coppia (m, c) e

- per ogni k_1 si calcola e si salva $\mathcal{C}_{DES}(m, k_1)$ in una tabella
- per ogni k_2 si calcola $\mathcal{D}_{DES}(c, k_2)$ e si cerca nella tabella

Quindi il costo è $O(2^b + 2^b) = O(2^{b+1}) \Rightarrow 2^{57}$. Essendo la sicurezza di soli 57 bit, si usa una composizione tripla a tre o a due chiavi, denominati:

3TDEA: $c = \mathcal{C}_{DES}(\mathcal{D}_{DES}(\mathcal{C}_{DES}(m, k_1), k_2), k_3)$
per ottenere il crittogramma, si cifra il messaggio con una chiave k_1 e si ottiene una certa stringa che viene decifrata con una chiave diversa k_2. In seguito si cifra nuovamente con una chiave k_3. Non si cifra tre volte per un fatto di retrocompatibilità con l'applicazione singola del DES, ovvero basta utilizzare tre chiavi uguali se non si vuole supportare la cifratura tripla. Anche in questa variante vi è un meet in the middle: ci si aspetta una sicurezza $3 \cdot 56 = 168$ bit ma in realtà è di soli 112 bit

2TDEA: $c = \mathcal{C}_{DES}(\mathcal{D}_{DES}(\mathcal{C}_{DES}(m, k_1), k_2), k_1)$
anche in questo caso vi è una sicurezza di soli 112 bit

6.3 AES

Il NIST nel 1998 pubblicò un bando per prendere in considerazione un nuovo cifrario simmetrico per le comunicazioni non classificate per il ventunesimo secolo. I requisiti del bando erano:

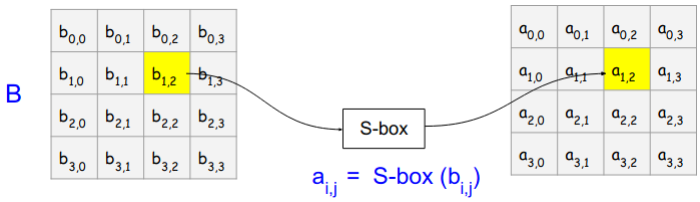
- *sicurezza*, resistenza a tutti gli attacchi crittoanalistici noti; correttezza del processo di cifratura e "casualità" dell'uscita
- *costo di realizzazione*, costo della realizzazione hardware; velocità di cifratura e decifrazione; occupazione di memoria e libero da brevetti
- *caratteristiche algoritmiche*, flessibile, portabile e applicabile a chiavi di $128, 192, 256$ bit

Il bando fu vinto dall'**AES** che è ad oggi lo standard per le comunicazioni riservate ma non classificate. È pubblicamente noto e realizzabile in hardware su computer di ogni tipo. AES permette di scegliere le chiavi di $128, 192$ o 256 bit. Il messaggio è diviso in blocchi lunghi come la chiave ed essa è utilizzata per trasformare un blocco del messaggio in un blocco del crittogramma. Come nel DES il processo di cifratura e decifrazione di AES opera per fasi. Ogni fase opera su un blocco di 128 bit, logicamente organizzato come una matrice bidimensionale B di 16 byte. La chiave iniziale è caricata, per colonne, in una matrice W di byte 4×4. Dato che nella variante da 128 bit vi sono 10 fasi, si espande il blocco W da 4 colonne a 44 colonne. Il gestore delle chiavi genera le sottochiavi di fase con questa legge:

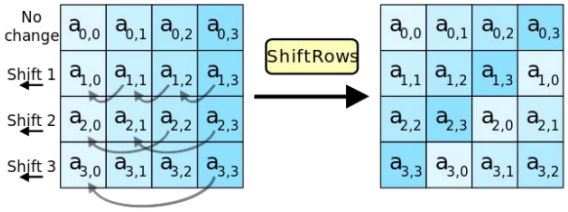
$$W[i] = \begin{cases} W[i-4] \oplus W[i-1] & \textit{se } i \textit{ non è un multiplo di } 4 \\ W[i-4] \oplus T(W[i-1]) & \textit{se } i \textit{ è un multiplo di } 4 \end{cases}$$

dove T è la S-box. La chiave $k(i)$ per la i-esima fase è data dalle 4 colonne $W[4i], W[4i+1], W[4i+2], W[4i+3]$. A grandi linee l'organizzazione è la seguente:

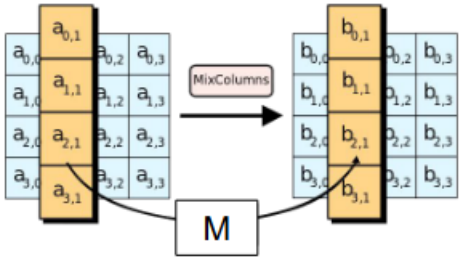
- **Trasformazione lineare**, il messaggio m è caricato nella matrice B. Ogni byte di B è posto in XOR con il corrispondente byte della chiave iniziale. Segue la fase sotto riportata, ripetuta per dieci volte
- **Organizzazione di una fase**, consiste nelle 4 operazioni:
  - *Substitution byte*
    ogni byte della matrice B è trasformato mediante una S-box. La S-box è una matrice 16×16 byte, che contiene una permutazione di tutti i 256 interi a 8 bit. La S-box si usa in forma tabellare per rendere la computazione veloce e ogni byte di B viene prima sostituito con il suo inverso moltiplicativo in $GF(2^8)$, e poi moltiplicato per una matrice 8×8 e sommato con un vettore colonna.



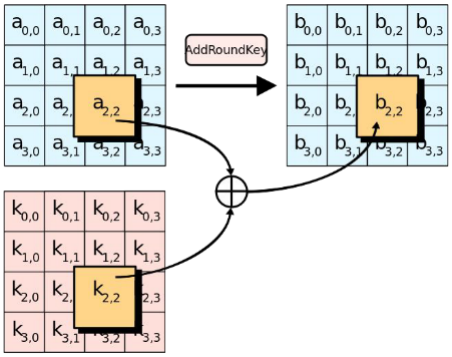
– *Shift Rows*
i byte di ogni riga della matrice vengono shiftati ciclicamente verso sinistra di $0, 1, 2$ e 3 posizioni, rispettivamente. In questo modo i 4 byte di ogni colonna si disperdono su 4 colonne diverse



– *Mix Columns*
ogni colonna del blocco viene moltiplicata per una matrice M di 4×4 byte. La moltiplicazione è eseguita $mod\ 2^8$, e la somma modulo 2. La matrice M è scelta in modo che ciascun byte della colonna venga mappato in un nuovo byte che è funzione di tutti e 4 byte presenti nella colonna. *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 risultante è posto in XOR con un byte della chiave locale di fase.



La decifrazione dell'AES, a differenza del DES, è simile alla cifratura ma non è uguale. Ad oggi nessun attacco è stato in grado di compromettere AES anche nella sua versione più semplice con chiave di 128 byte. Tutti i bit della chiave sono bit di sicurezza. Esistono attacchi più efficienti di un attacco esauriente sulle chiavi per le versioni di AES con 6 fasi, ma nessun attacco è più efficiente se le fasi sono almeno 7. Per AES, come praticamente per tutti i cifrari a chiave simmetrica, si conoscono pericolosi attacchi **side-channel** che non sfruttano le caratteristiche del cifrario ma le possibili debolezze della piattaforma su cui esso è implementato.

6.4 Cifrari a composizione di blocchi

I cifrari simmetrici impiegati in pratica funzionano a blocchi, in particolare di 64 bit nel DES e 128 bit nell'AES (se il messaggio non ha una dimensioni multipla della dimensione del blocco si utilizza il *padding*). Ciò espone la comunicazione ad attacchi fin qui non considerati: blocchi uguali nel messaggio producono blocchi cifrati uguali; poca

diffusione e periodicità nel crittogramma utile per la crittoanalisi. Per ovviare il problema si utilizza la composizione di blocchi con un metodo chiamato **Cipher Block Chaining**. Con il CBC, il cifrario risultante è ancora strutturato a blocchi, ma blocchi uguali nel messaggio vengono cifrati in modo diverso eliminando così la periodicità. Il metodo si impiega in cifratura e in decifrazione utilizzando le funzioni $\mathcal{C}$ e $\mathcal{D}$ del cifrario d'origine

Cifratura $c_i = \mathcal{C}(m_i \oplus c_{i-1}, k)$
dove m_i è un blocco del messaggi e c_{i-1} è il crittogramma del blocco precedente. Serve una sequenza c_0 di b bit diversa per ogni messaggio per innestare tutto il procedimento.

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

Il processo di cifratura è strettamente sequenziale in quanto il calcolo di ogni c_i impiega il risultato c_{i-1} del passo precedente, mentre il processo di decifrazione può essere eseguito in parallelo se tutti i blocchi cifrati sono disponibili. La sequenza c_0 e la sua modifica per ogni messaggio sono necessarie per evitare che messaggi uguali generino crittogrammi uguali. Essa può essere costruita in modo pseudocasuale e viene solitamente spedita in chiaro. Tutto ciò non pregiudica in alcun modo la robustezza del sistema crittografico. Tra le proprietà più interessanti del CBC vi è quella di **resistenza agli errori** nel crittogramma, dovuti per esempio alla trasmissione o a manomissione controllate da un crittoanalista, poiché la sostituzione di un bit nel blocco c_i induce un errore nella decifrazione dell'intero blocco m_i e del corrispondente bit nel blocco m_{i-1}, ma qui la propagazione dell'errore si arresta.

6.5 Altri cifrari a chiave segreta

Brevemente:

- **RC5** è abbastanza simile al DES migliorando alcune parti. È composto da blocchi di 64 bit e la chiave la si può scegliere come si vuole ma deve essere multiplo di 32. Ci sono r fasi e usa shift ciclico, XOR e addizione $mod\ 2^{32}$. L'RC5 è molto veloce, resiste con successo agli attacchi standard se c e r sono scelti bene, è sicuro e impiegato con una certa frequenza.
- **IDEA**, utilizza chiavi da $128\ bit$ e le solite operazioni di shift ciclico, XOR, moltiplicazione $mod(2^{16}+1)$ e addizione $mod\ 2^{16}$. È più semplice e sicuro del DES, e la sicurezza poggia su basi teoriche molti forti. Rimasto assolutamente inviolato ma non è diventato il nuovo standard

Chapter 7

Crittografia a chiave pubblica

7.1 Il problema dello scambio delle chiavi

Nei cifrari *simmetrici* visti sinora la chiave di cifratura è uguale a quella di decifrazione (o comunque ciascuna può essere facilmente calcolata dall'altra), ed è nota solo ai due partner che la scelgono di comune accordo e la mantengono segreta. Il problema è come scambiarsi la chiave segreta con facilità e sicurezza? La chiave serve per comunicare in sicurezza, ma deve essere stabilita comunicando "in sicurezza" senza poter usare il cifrario

Problema. N utenti vogliono comunicare tra loro su un canale condiviso. Charlie può vedere i messaggi scambiati da Alice e Bob, ma non li può decifrare.

Soluzione naïve: ogni utente memorizza $N-1$ chiavi diverse, una chiave per ogni coppia di utenti, condivise con ciascun altro utente

Soluzione alternativa: ricorrere ad un **trusted 3rd party** (TTP): ogni utente si deve ricordare una sola chiave per comunicare con TTP e TTP gestisce la creazione delle chiavi condivise tra le coppie di utenti. Esempio: Alice vuole comunicare con Bob usando un TTP, che conosce la chiave k_A di Alice e la chiave k_B di Bob

- Alice avvisa TTP: "Voglio comunicare con Bob"
- TTP genera casualmente una chiave k_{AB}
- TTP manda ad Alice $c_A = \mathcal{C}(k_{AB}, k_A)\ and\ c_B = \mathcal{C}(k_{AB}, k_B)$
- Alice invia c_B a Bob
- Alice e Bob iniziano a comunicare usando la chiave k_{AB}

Questo schema presenta due problemi principali: TTP deve essere sempre online e TTP conosce tutte le chiavi. È un unico punto di errore per l'intero sistema. Ci sono situazioni in cui questo approccio ha senso, come per esempio, all'interno di una università o un ambiente ristretto

Nel 1976 viene proposta alla comunità scientifica un algoritmo per generare e scambiare una chiave segreta su un canale insicuro senza la necessità che le due parti si siano scambiate informazioni o incontrate in precedenza. Questo algoritmo, detto **protocollo DH** è ancora largamente usato nei protocolli crittografici su Internet. Nel 1976 Diffie e Hellman, propongono alla comunità scientifica anche la definizione di **crittografia a chiave pubblica** con l'obiettivo di permettere a tutti di inviare messaggi cifrati ma abilitare solo il ricevente (Bob) a decifrarli

7.2 Cifrari a chiave pubblica

Nei cifrari a chiave pubblica, o *asimmetrici*, le chiavi di cifratura e di decifrazione sono completamente diverse

k_{pub} per cifrare: è pubblica, nota a tutti

k_{priv} per decifrare: è privata, nota solo a Bob

In sostanza esiste una coppia $\langle k_{pub}, k_{priv} \rangle$ per ogni utente del sistema, scelta da questi nella sua veste di possibile destinatario. Quindi:

La *cifratura* di un messaggio m da inviare a Bob è eseguita da qualunque mittente come $c = \mathcal{C}(m, k_{pub})$. La chiave k_{pub} e funzione di cifratura sono note a tutti

La *decifrazione* è eseguita da Bob come $m = \mathcal{D}(c, k_{priv})$. La funzione di decifrazione è nota a tutti, ma k_{priv} non è disponibile agli altri

Per funzionare correttamente, il processo di cifratura e decifrazione deve soddisfare alcune proprietà:

1. Bob deve avere la possibilità di interpretare qualunque messaggio che gli altri utenti decidano di spedirgli, quindi per ogni possibile messaggio m si ha $\mathcal{D}(\mathcal{C}(m, k_{pub}), k_{priv}) = m$
2. Efficienza e sicurezza del sistema, più esattamente:
   a) la coppia di chiavi è *facile* da generare, e deve risultare praticamente impossibile che due utenti scelgano la stessa chiave (**generazione casuale delle chiavi)**
   b) dati m e k_{pub} è *facile* per Alice calcolare il crittogramma $c = \mathcal{C}(m, k_{pub})$ (**adottabilità del sistema**)
   c) dati c e k_{priv} è *facile* per Bob calcolare il messaggio originale $m = \mathcal{D}(c, k_{priv})$ (**adottabilità del sistema**)
   d) pur conoscendo il crittogramma c, la chiave pubblica, e le funzioni di cifratura e decifrazione, è *difficile* per il crittoanalista risalire al messaggio m (**sicurezza del cifrario**)

La funzione di cifratura deve avere una **funzione one-way**, cioè facile da calcolare e difficile da invertire, ma deve contenere un meccanismo segreto detto **trap-door** che ne consente la facile invertibilità solo a chi conosca tale meccanismo. La crittografia a chiave pubblica non ha sostituito la crittografia simmetrica, ma solo affiancata. Diffie e Hellman quando hanno elencato le proprietà di cifratura e di decifrazione non sono riusciti a trovare una funzione one-way-trap-door adatta. Nel 1977 Rivest, Shamir e Adleman hanno trovato la funzione one-way-trap-door.

7.3 Alcuni richiami di algebra modulare

Molti logaritmi crittografici utilizzano l'algebra modulare essenzialmente per due motivi. Il primo è ridurre lo spazio dei numeri su cui gli algoritmi sono chiamati a operare e quindi aumentare la velocità di calcolo; il secondo è rendere difficile alcuni problemi computazionali che sono semplici nell'algebra non modulare. Si dice che a è congruo a b modulo n, scritto $a \equiv b \; mod \; n$ con $a, b \geq 0$ e $n > 0$, se e solo se esiste k intero tale che $a = b + kn$. Proprietà:

- $(a + b) \; mod \; m = (a \; mod \; m + b \; mod \; m) \; mod \; m$
- $(a - b) \; mod \; m = (a \; mod \; m - b \; mod \; m) \; mod \; m$
- $(a \times b) \; mod \; m = (a \; mod \; m \times b \; mod \; m) \; mod \; m$
- $a^{r \times s} \; mod \; m = (a^r \; mod \; m)^s \; mod \; m$ con r e s interi positivi

Preso un numero intero positivo n si indica con $\mathcal{Z}_n = \{0, 1, ..., n-1\}$ l'insieme di tutti gli interi non negativi minori di n, e con $\mathcal{Z}_n^*$ l'insieme di tutti gli elementi di $\mathcal{Z}_n$ co-primi con n. Se n è primo, $\mathcal{Z}_n^* = \{1, 2, ..., n-1\}$. Se n non è primo, calcolare $\mathcal{Z}_n^*$ è computazionalmente difficile. La **funzione di Eulero** è definita come $\phi(n) = |\mathcal{Z}_n^*|$. Teoremi:

- Se n è primo $\Rightarrow \phi(n) = n - 1$
- Se n è prodotto di due primi $\Rightarrow \phi(n) = (p-1)(q-1)$
- Per $n > 1$ e per ogni a **primo** con $n \Rightarrow a^{\phi(n)} \equiv 1 \; mod \; n$ (**teorema di Eulero**)
- Per n primo e per ogni $a \in \mathcal{Z}_n^* \Rightarrow a^{n-1} \equiv 1 \; mod \; n$ (**piccolo teorema di Fermat**)

L'**inverso** in modulo si può calcolare come $a^{-1} = a^{\phi(n)-1} \; mod \; n$. Esistenza ed unicità dell'inverso sono garantiti se e solo se a e n sono coprimi, ovvero, $mcd(a, n) = 1$. L'equazione $ax \equiv b \; mod n$ ammette soluzione se e solo se $mcd(a, n)$ divide b, in questo caso si hanno esattamente $mcd(a, n)$ soluzioni distinte. L'equazione $ax \equiv b \; mod \; n$ ammette *un'unica soluzione* se e solo se a e n sono coprimi, quindi se e solo se esiste l'inverso a^{-1} di a. **Teorema cinese del resto**: siano $n_1, ..., n_k$ interi a due a due comprimi, allora, comunque si scelgano degli interi $a_1, ..., a_k$ *esiste un'unica soluzione* intera x del sistema di congruenze. Quest'ultimo teorema mostra come, sotto opportune condizioni, un sistema di congruenze possa essere sostituito da un'unica congruenza. $a \in \mathcal{Z}_n^*$ è un **generatore** di $\mathcal{Z}_n^*$ se la funzione $a^k \; mod \; n$, con $1 \geq k \geq \phi(n)$, genera tutti e soli gli elementi di $\mathcal{Z}_n^*$. Produce come risultati tutti gli elementi di $\mathcal{Z}_n^*$ ma in un ordine difficile da prevedere. Esempio: $g = 2$ è un generatore di $Z_{13}^* = \{1, 2, ..., 12\}$

x	1	2	3	4	5	6	7	8	9	10	11	12
2^x	2	4	8	3	6	12	11	9	5	10	7	1

Per il teorema di Eulero si conosce che 1 si ottiene quando si eleva per $\phi(n)$, quindi 1 è generato sempre alla fine della sequenza, ovvero per $k = \phi(n)$. Non per tutti i valori di n, $\mathcal{Z}_n^*$ ha generatori, per esempio $\mathcal{Z}_8^*$ non ammette generatori. Se n è un numero primo, $\mathcal{Z}_n^*$ ha almeno un generatore. Per n primo, non tutti gli elementi di $\mathcal{Z}_n^*$ sono suoi generatori. Per n primo, i generatori di $\mathcal{Z}_n^*$ sono in totale $\phi(n-1)$

Esempio. 3 genera $\mathcal{Z}_7^*$
$\mathcal{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$
$\phi(7) = 6$
$3^k \ mod \ 7 = 3, 2, 6, 4, 5, 1 \qquad 1 \leq k \leq 6$

Esempio. 2 **non** genera $\mathcal{Z}_7^*$
ce ne si può accorgere perché $2^3 \ mod \ 7 = 1$
ma da Eulero si è visto che deve generare 1
soltanto quando $k = \phi(n)$

Un problema rilevante sui generatori in crittografia è il **calcolo del logaritmo discreto**: risolvere nell'incognita x l'equazione $a^x \equiv b \ mod \ n$ con n primo. L'equazione ammette una soluzione per ogni valore di b se e solo se a è un generatore di $\mathcal{Z}_n^*$. Tuttavia non è noto a priori in che ordine sono generati gli elementi $\mathcal{Z}_n^*$, quindi non è noto per quale valore di x si genera $b \ mod \ n$

7.4 Le funzioni one-way trap-door

Il calcolo delle funzioni one-way trap-door è facile, ma invertire tale funzioni è difficile (non si conosce algoritmo polinomiale), a meno che non si dispongono di informazioni aggiuntive. Si considerano ora tre di queste funzioni particolarmente rilevanti in crittografia:

- **Fattorizzazione**, calcolare il prodotto n di due interi p e q è facile, richiede tempo quadratico nella lunghezza della loro rappresentazione. Invertire la funzione, cioè ricostruire p e q a partire da n richiede tempo esponenziale (anche se non vi è dimostrazione del fatto che il problema sia NP hard). Trap door: se si conosce uno dei fattori (la chiave segreta) ricostruire l'altro è facile
- **Calcolo della radice in modulo**, calcolare la potenza $y = x^z \ mod \ s$, con x, z, s intero, richiede tempo polinomiale se si procede per successive esponenziazioni, si eseguono $\Theta(log_2 z)$ moltiplicazioni. Se s non è primo, invertire la funzione $x = y^{1/z} \ mod \ s$ richiede tempo esponenziale per quanto noto
- **Calcolo del logaritmo discreto**, calcolare la potenza $y = x^z \ mod \ s$ è facile. Invertire rispetto a z, cioè trovare z tale che $y = x^z \ mod \ s$ dati x, y, s è computazionalmente difficile.

7.5 Pregi e difetti del nuovo metodo

Rispetto all'impiego di cifrari simmetrici, la crittografia a chiave pubblica presenta due vantaggi immediati e strettamente connessi:

- se gli utenti di un sistema sono n, il numero complessivo di chiavi (pubbliche e private) è $2n$ anzichè $n(n-1)/2$;
- non è richiesto alcun scambio segreto di chiavi tra gli utenti

Accanto a questi vantaggi i cifrari a chiave pubblica comportano alcuni svantaggi altrettanto immediati:

- sono sistemi molto più lenti di quelli basati sui cifrari simmetrici
- sono esposti ad attacchi di tipo *chosen plain-text*. Un crittoanalista può scegliere un numero qualsiasi di messaggi in chiaro $m_1, ..., m_h$ e cifrarli utilizzando la funzione pubblica $\mathcal{C}$ e la chiave pubblica $k[pub]$ del destinatario, ottenendo i crittogrammi $c_1, ..., c_h$. A questo punto, spiando sul canale di comunicazione, egli può confrontare qualsiasi messaggio cifrato c^* che viaggia verso il destinatario con i crittogrammi in suo possesso: se c^* coincide con uno di essi il messaggio è automaticamente decifrato; se invece $c^* \neq c_i$, per ogni i, il crittoanalista ha comunque acquisito una informazione importante, cioè che il messaggio è diverso da quelli che lui ha scelto. L'attacco è particolarmente pericoloso se il crittoanalista sospetta che il destinatario debba ricevere un messaggio particolare ed è in attesa di vedere quando questo accada

Poiché i cifrari simmetrici e asimmetrici presentano pregi e difetti complementari, nei protocolli crittografici esistenti si tende ad adottare un **approccio ibrido**: si usa un cifrario a chiave segreta (AES) per le comunicazioni di massa e un cifrario a chiave pubblica per scambiare le chiavi segrete relative al primo, senza incontri fisici tra gli utenti. In questo modo la trasmissione dei messaggi lunghi avviene ad alta velocità mentre è lento lo scambio delle chiavi segrete che però sono assai più brevi. Inoltre l'attacco *chosen plain-text* di cui soffrono i cifrari simmetrici è ovviato se l'informazione cifrata con la chiave pubblica (chiave segreta dell'AES) è scelta in modo da risultare imprevedibile al crittoanalista. La chiave pubblica deve essere estratta da un certificato digitale valido, per evitare attacchi *man-in-the-middle*. Alla definizione del nuovo protocollo a chiave pubblica seguirono in breve tempo varie realizzazione:

- **Merkle**, basava la difficoltà di inversione della funzione $\mathcal{C}$ sulla risoluzione del problema dello zaino. Il cifrario è stato violato, mostrando ancora una volta con quanta cautela vada affrontato il problema della sicurezza. I successivi cifrari basati sullo stesso problema sono invece rimasti inviolati
- **RSA**, fonda la sua sicurezza sulla difficoltà di fattorizzare grandi numeri interi. Benché tale problema non sia dimostratamente NP hard, e quindi potrebbe essere "più semplice" del problema dello zaino, RSA è ad oggi inviolato, ed è stato il primo cifrario asimmetrico di largo impiego
- **ElGamal**, fonda la sua sicurezza sulla difficoltà di calcolare il logaritmo discreto

7.6 Il cifrario RSA

RSA si basa sulle moltiplicazioni di due numeri primi p e q: calcolare $n = p \times q$ è facile ma invertire la funzione, cioè ricostruire p e q a partire da n richiede tempo esponenziale. La trap door è la funzione di Eulero o il fatto che se si conosce uno dei fattori allora ricostruire l'altro è facile. Si studia l'organizzazione generale:

- **Creazione della chiave**, ogni utente destinatario esegue le seguenti operazioni:
  - sceglie due numeri primi p e q molto grandi, ovvero, tali che $n = p \times q$ abbia almeno 2048 cifre binarie per protezione dati fino al 2030, e 3072 cifre binarie per protezione dati oltre il 2030. Richiede tempo polinomiale (Miller-Rabin)
  - calcola $n = p \times q$, e la funzione di Eulero $\phi(n) = (p-1)(q-1)$. Richiede tempo polinomiale
  - sceglie un intero e tale che $e < \phi(n)$ e $mcd(e, \phi(n)) = 1$. Richiede tempo polinomiale
  - calcola l'intero d inverso di e $\boldsymbol{mod\ \phi(n)}$, d esiste ed è unico perché $mcd(e, \phi(n)) = 1$
  - rende pubblica la chiave $k[pub] = \langle e, n \rangle$, e mantiene segreta la chiave $k[priv] = \langle d \rangle$
- **Messaggio**, il messaggio m è una sequenza binaria trattata come un intero e deve essere $m < n$, il che è sempre possibile dividendo il messaggio in blocchi $b = \lfloor log_2 n \rfloor$ bit. Nella pratica si stabilisce un limite comune per impiegare blocchi delle stesse dimensioni per tutti i destinatari: $m < 2^b < n$
- **Cifratura**, si calcola come $c = m^e \ mod\ n$. Richiede tempo polinomiale
- **Decifrazione**, si decifra calcolando $m = c^d \ mod\ n$. Richiede tempo polinomiale

Esempio. Si scelgono $p = 5$ e $q = 11$, allora $n = p \times q = 5 \times 11 = 55$ e $\phi(n) = (p-1)(q-1) = 4 \times 10 = 40$
Si determina un e minore e comprimo con $\phi(n)$: $e = 7$
Si calcola l'inverso d: $7^{-1} \ mod\ 40$

	40	7	
40	1	0	
7	0	1	
$40 - 5 \times 7 = 5$	1	-5	*moltiplica la riga del 7 per -5 e poi somma I riga + II riga*
$7 - 1 \times 5 = 2$	-1	6	*moltiplica la riga del 5 per -1 e poi somma II riga + III riga*
$5 - 2 \times 2 = 1$	3	-17	*moltiplica la riga del 2 per -2 e poi somma III riga + IV riga*

L'inverso di $7 \ mod\ 40$ è -17 che per valori positivi è 23. Riprova: $23 \cdot 7 = 161 \ mod\ 40 = 1$ OK
Si pubblica $k[pub] = \langle 7, 55 \rangle$ e si mantiene segreta $k[priv] = \langle 23 \rangle$
Chiunque vuole spedire un messaggio $m < 55$ dovrà calcolare $m^7 \ mod\ 55$ e recapitare c.
Per decifrare c si deve calcolare $m = c^{23} \ mod\ 55$

Dimostrazione. Per dimostrare la correttezza del RSA si dimostrare che $(m^e \ mod \ n)^d \ mod \ n = m$. che può essere riscritta come $m^{ed} \ mod \ n = m$. Per la dimostrazione si distinguono due casi, relativi ai valori di p e q scelti dal destinatario e al valore di m scelto dal mittente

1. p e q non dividono m. Si ha $mdc(m, n) = 1$ per il teorema di Eulero risulta $m^{\phi(n)} \equiv 1 \ mod \ n$. Poichè d è l'inverso di e si ha $e \times d \equiv 1 \ mod \ \phi(n)$, ovvero $e \times d = 1 + r\phi(n)$ con r intero positivo opportuno. Si ottiene quindi $m^{ed} \ mod \ n = m^{1+r\phi(n)} \ mod \ n = m \times (m^{\phi(n)})^r \ mod \ n = m \times 1^r \ mod \ n = m$
2. p divide m oppure q divide m, ma non entrambi. Si suppone che p divide m (e q non divide m) allora $m \equiv m^r \equiv 0 \ mod \ p$, per qualunque intero positivo r. Come r si sceglie $r = e \times d$ allora $m^{ed} - m \equiv 0 \ mod \ p$. Siccome q non divide m, si calcola $m^{ed} \ mod \ q = m^{1+r\phi(n)} \ mod \ q = m \times m^{r(p-1)(q-1)} \ mod \ q = m \times (m^{(q-1)})^{r(p-1)} \ mod \ q = m(m^{\phi(q)})^{r(p-1)} \ mod \ q = m \times (1)^{r \times (p-1)} \ mod \ q = m \ mod \ q$. Quindi si è trovato che $m^{ed} - m \equiv 0 \ mod \ q$. Ne consegue che $m^{ed} - m$ è divisibile sia per p che per q, quindi è divisibile per il loro prodotto $p \times q = n$; ovvero $(m^{ed} - m) \equiv 0 \ mod \ n$

7.6.1 Sicurezza

La sicurezza del cifrario è strettamente legata alla difficoltà di fattorizzare un numero intero arbitrario molto grande. Infatti la ricostruzione degli interi p, q, il cui prodotto costituisce il valore n dichiarato in una chiave pubblica, permette di calcolare immediatamente $\phi(n)$ e quindi la chiave privata di chi ha pubblicato n. In sostanza la facoltà di fattorizzare efficientemente un intero implica quella di forzare efficientemente il cifrario ma non è nota l'implicazione inversa, cioè non si sa se forzare efficientemente il cifrario implichi fattorizzare efficientemente un intero. Infatti nonostante numerosi studi non è ancora noto se per forzare l'RSA si debba crucialmente passare attraverso la fattorizzazione di n anche se ciò è ritenuto molto plausibile.

- Per decifrare un crittogramma c sarebbe sufficiente calcolare la sua radice e-esima modulo n, perché tutti conoscono $k[pub] = \langle e, n \rangle$ e sanno che $c = m^e \ mod \ n$. D'altra parte il calcolo della radice e-esima nell'algebra modulare può essere eseguito efficientemente se il modulo n è primo, ma è ritenuto difficile quanto la fattorizzazione se n è composto. Ne segue che il calcolo di m a partire da e, n, c non è più facile del calcolo dei fattori primi di n: tanto vale quindi calcolare questi ultimi, anche perché ciò implica la forzatura totale del sistema e non la sola decifrazione di un particolare crittogramma.
- Un metodo alternativo di forzatura potrebbe basarsi sulla scoperta di $\phi(n)$ senza passare attraverso i valori p, q poiché conoscendo $\phi(n)$ si calcola immediatamente la chiave segreta. Ma ciò non cambia le cose: si può infatti scrivere $\phi(n) = (p-1)(q-1) = n - (p+q) + 1$ e, conoscendo $\phi(n)$ si può calcolare $x_1 = (p+q)$. Osservando poi che vale l'uguaglianza: $(p-q)^2 = (p+q)^2 - 4n$ si può calcolare $x_2 = (p-q)$ da cui infine si ha $p = (x_1 + x_2)/2, q = (x_1 - x_2)/2$. Ciò significa che il calcolo di $\phi(n)$ e la fattorizzazione di n sono problemi computazionalmente equivalenti poiché ciascuno si trasforma nell'altro in tempo polinomiale.
- Infine si potrebbe condurre un attacco esauriente sulla chiave segreta d, cioè verificare, per ogni possibile valore di d, se la decifrazione dà origine a un messaggio significativo. Il processo potrebbe però essere molto più costoso della fattorizzazione di n poiché, in dipendenza dalla scelta di e, il valore di d può essere molto più grande di quelli di p e di q. Non è noto se sia possibile calcolare efficientemente d utilizzando solo la chiave pubblica $\langle e, n \rangle$ ed eventualmente il crittogramma c.

In sostanza tutti i tentativi compiti hanno confermato la congettura che per forzare il cifrario RSA implichi fattorizzare n. La fattorizzazione è un problema difficile, ma non più come un tempo: da un lato la potenza di calcolo aumenta, dall'altro gli algoritmi di fattorizzazione vengono raffinati. Esistono algoritmi relativamente veloci di complessità subesponenziale, come per esempio il **General Number Field Sieve** che richiede $O(2^{\sqrt{b \ log \ b}})$ operazioni per fattorizzare un intero n con $[log_2 n] + 1$ bit. Un attacco bruteforce ne richiede $O(n)$ ovvero $O(2^{log \ n})$. Con la potenza di calcolo attuale, usando l'algoritmo GNFS è possibile fattorizzare semiprimi fino a circa 768 bit. Per interi con una struttura particolare, esistono algoritmi di fattorizzazione particolarmente efficienti:

- sia $p-1$ che $q-1$ devono contenere un fattore primo grande, altrimenti n si fattorizza velocemente
- $mcd(p-1, q-1)$ deve essere piccolo, conviene scegliere p e q tale che $(p-1)$ e $(q-1)$ siano coprimi
- non riusare uno dei primi per altri moduli
- scegliere p e q non troppo vicini tra loro

Esponenti e e d bassi sono attraenti perché accelerano cifratura o decifrazione ma d dovrebbe essere scelto sufficientemente grande per evitare attacchi bruteforce. Attenzione: se m ed e sono così piccoli che $m^e < n$ allora risulta facile trovare la radice e-esima di c poiché $c = m^e$, non interviene la riduzione in modulo. I valori di e da evitare sono $\phi(n)/2 + 1$ e $\phi/k + 1$ se k divide $p - 1$ e $q - 1$ altrimenti $m^e \ mod \ n = m$ quando m e n sono primi tra loro.

7.6.2 Attacchi con lo stesso valore di e

Si suppone che esistano almeno e utenti che hanno scelto lo stesso valore di e e che almeno e fra essi ricevono lo stesso messaggio m attraverso i crittogrammi $c_1 = m^e \ mod \ n_1$, $c_2 = m^e \ mod \ n_2, ..., c_e = m^e \ mod \ n_e$ dove $\forall \ i, j \ . \ 1 \leq i < j \leq e \ . \ mcd(n_i, n_j) = 1$. Si è detto che tutti gli utenti hanno ricevuto lo stesso messaggio, quindi m per essere cifrato con le chiavi pubbliche deve soddisfare $\forall i \ . \ 1 \leq i \leq e \ . \ m < n_i$. Con queste condizioni il crittoanalista può applicare il teorema cinese del resto che li consente di calcolare in tempo polinomiale un valore m', minore di n tale che $m' \equiv m^e \ mod \ n$ dove $n = \prod_{i=1}^{e} \ n_i$

$$
\begin{array}{rcll}
m' & \equiv & m^e \ mod \ n & \text{per definizione di congruenza} \\
m' \ mod \ n & \equiv & m^e \ mod \ n & m' \ mod \ n = m' \text{ perché } m' < n \\
m' & \equiv & m^e \ mod \ n & m^e = \underbrace{m \cdot m ... m}_{\text{e volte}} < n_1 \cdot n_2 ... n_e = n \text{ cioè } \forall m < n_i \\
m' & = & m^e &
\end{array}
$$

quindi la congruenza diventa una uguaglianza $m' = m^e$ dunque $m = \sqrt[e]{m'}$. Per essere al sicuro da questi attacchi si evita di mandare a utenti diversi che hanno lo stesso valore di e lo stesso messaggio, per fare ciò basta aggiungere una sequenza casuale di bit diversa per ogni destinatario alla fine di ogni messaggio: *padding*

7.6.3 Common modulus attack

Vi sono due utenti u_1, u_2 con chiavi pubbliche rispettivamente $\langle e_1, n \rangle$ e $\langle e_2, n \rangle$ con $mcd(e_1, e_2) = 1$ il che è molto probabile se e_1 e e_2 sono scelti in modo casuale. Si suppone ora che i due utenti ricevono lo stesso messaggio cifrato con la propria chiave pubblica: l'utente u_1 riceve $c_1 = m^{e_1} \ mod \ n$ e l'utente u_2 riceve $c_2 = m^{e_2} \ mod \ n$. Siccome e_1 e e_2 sono coprimi, il crittoanalista sa che devono esistere (e si calcolano in tempo polinomiale) r, s tali che $r \cdot e_1 + s \cdot e_2 = 1$. Per fare 1 significa che uno tra r e s deve essere minore di 0. Si pone che $r < 0$ (il caso $s < 0$ è simmetrico). Il crittoanalista conosce c_1, c_2, e_1, e_2, n e si è calcolato r e s.

$$
\begin{array}{rcll}
m & = & m^1 & \\
 & = & m^{r \cdot e_1 + s \cdot e_2} & \text{dato che } m < n \text{ si può aggiungere il modulo} \\
 & = & m^{r \cdot e_1 + s \cdot e_2} \ mod \ n & \text{per proprietà potenze} \\
 & = & (m^{r \cdot e_1} \ mod \ n)(m^{s \cdot e_2} \ mod \ n) \ mod \ n & \text{per proprietà potenze} \\
 & = & (m^{e_1} \ mod \ n)^r (m^{e_2} \ mod \ n)^s \ mod \ n & m^{e_1} \ mod \ n = c_1 \quad m^{e_2} \ mod \ n = c_2 \\
 & = & c_1^r \cdot c_2^s \ mod \ n &
\end{array}
$$

Siccome $s > 0$, $c_2^s \ mod \ n$ si calcola in tempo polinomiale con l'algoritmo di esponenziazione veloce. Per quanto riguarda $c_1^r \ mod \ n$ lo si può riscrivere come $c_1^r \ mod \ n = (c_1^{-1})^{-r} \ mod \ n$ e se $mcd(c_1, n) = 1$ allora il crittoanalista calcola $c_1^{-1} \ mod \ n$ e $(c_1^{-1})^{-r}$, infine ottiene m come $m = (c_1^{-1})^{-r}(c_2)^s \ mod \ n$

7.6.4 Attacchi a tempo

Si basano sul tempo di esecuzione dell'algoritmo di decifrazione. Il crittoanalista cerca di avere informazioni sulla chiave privata controllando quanto è il tempo impiegato per la decifrazione. La decifrazione usa l'algoritmo di esponenziazione veloce che esegue una moltiplicazione ad ogni iterazione, più un'ulteriore moltiplicazione modulare per ciascun bit uguale a 1 in d. Il rimedio a questo attacco è aggiungere del ritardo casuale per confondere l'attaccante

7.7 Cifrari ibridi e scambio di chiavi

Poiché i cifrari asimmetrici sono esposti ad attacchi chosen plain-text e la loro realizzazione richiede tempi di cifratura e decifrazione molto maggiori di quelli dei cifrari simmetrici, i due tipi di cifrari vengono in genere utilizzati in combinazione dando origine a **cifrari ibridi** in cui un cifrario a chiave pubblica è utilizzato per lo scambio della chiave segreta che viene impiegato nelle successive comunicazioni simmetriche. Si studia la classica combinazione tra RSA e AES: la chiave segreta trasmessa con l'RSA, denominata *chiave di sessione* e indicata con $k[session]$, può essere modificata in ogni nuovo interscambio di messaggi senza che i due partner impegnati nella comunicazione

debbano incontrarsi di persona. I due partner potranno così utilizzare la potenza dei sistemi asimmetrici per lo scambio a distanza di chiavi segrete, che resisteranno ad attacchi chosen-plain-text se "prive di semantica". Inoltre la loro comunicazione non soffrirà della lentezza propria dell'RSA in quanto questo cifrario sarà utilizzato molto di rado e le comunicazioni saranno brevi perché lo sono le chiavi. In sintesi questo protocollo scambia i messaggi segreti nella forma $\langle C_{RSA}(k[session], k[pub]), C_{AES}(m, k[session])\rangle$. Il destinatario decifra il primo crittogramma con la sua chiave privata e trova $k[session]$, poi decifra il secondo crittogramma usando la chiave $k[session]$. Nel sistema ibrido RSA/AES il mittente ha il compito di generare la chiave di sessione e inviarla al destinatario prima del messaggio vero e proprio. perché ciò sia possibile il mittente deve disporre di risorse computazionali sufficienti a garantire la creazione di una chiave sicura, il che non è banale. Inoltre si preferisce attribuire pari responsabilità al mittente e al destinatario che potrebbero non conoscersi e quindi non fidarsi l'uno dell'altro. Per risolvere questi problemi si sostituisce l'RSA con il **protocollo DH**

7.8 Protocollo DH

L'algoritmo sfrutta la proprietà di funzione one-way del logaritmo discreto. I due partner generano la chiave $k[session]$ in modo *incrementale* inviandosi in chiaro alcuni pezzi di essa; questi pezzi sono sufficienti a ricostruire la chiave stessa solo se vengono combinati con le informazioni segretamente in possesso di ciascun partner e diverse per entrambi. In particolare:

1. Alice e Bob si accordano pubblicamente su un numero primo p molto grande, tipicamente di un migliaio di cifre binarie, e su un generatore g per $\mathcal{Z}_p^*$ con $g < p$. Se Alice e Bob non sono in grado di generare una coppia $\langle p, g\rangle$ possono utilizzare una coppia pubblica messa a disposizione nel sistema, che può essere la stessa per tutti gli utenti senza inficiare la sicurezza del protocollo.
2. Alice si sceglie a caso un intero a positivo con $1 < a < p-1$, calcola $A = g^a \ mod \ p$ e spedisce in chiaro questo valore a Bob
3. Bob estrae un intero positivo casuale b positivo con $1 < b < p-1$, calcola $B = g^b \ mod \ p$ e spedisce in chiaro questo valore ad Alice
4. Alice riceve B e calcola $k[session] = B^a \ mod \ p = g^{ba} \ mod \ p$ sfruttando la sua conoscenza privata di a
5. Bob riceve A e calcola $k[session] = A^b \ mod \ p = g^{ab} \ mod \ p$ sfruttando la sua conoscenza privata di b

Alla fine del protocollo entrambi i partner hanno generato la stessa chiave di sessione che viene così utilizzata per le cifrature simmetriche successive.

7.8.1 Attacco passivo/attivo

Un crittoanalista *passivo* può aver intercettato i valori p, g, A, B scambiati in chiaro tra i due partner, ma per calcolare la chiave di sessione deve risolvere l'equazione $A = g^a \ mod \ p$ rispetto a a, oppure $B = g^b \ mod \ p$ rispetto a b, che sono calcoli computazionalmente difficili. Tuttavia, il protocollo è vulnerabile agli attacchi *attivi* del tipo man-in-the-middle. Il crittoanalista prende il nome di Eve e sceglie un intero qualsiasi z, calcola il valore $Z = g^z \ mod \ p$ e si frappone sul canale bloccando le comunicazioni tra Alice e Bob per sostituirle con le proprie. Eve cattura i messaggi A e B di Alice e Bob, e risponde a entrambi con Z. Alice e Bob interpretano Z come proveniente dall'altro partner e costruiscono le chiavi $K_A = Z^a \ mod \ p = g^{az} \ mod \ p$ e $K_B = Z^b \ mod \ p = g^{bz} \ mod \ p$ con cui proseguiranno la comunicazione con Eve che colloquia con Alice usando K_A e con Bob usando K_B

7.9 Il cifrario di ElGamal

Il cifrario di ElGamal ha lo stesso grado di sicurezza dell'RSA e si basa sulla difficoltà di calcolare i logaritmi discreti. Passaggi:

- **Creazione della coppia di chiavi**
  si sceglie un numero primo p, un generatore g di Z_p^* e si sceglie a caso un intero $x \in [2, p-2]$. In seguito si calcola $y = g^x \ mod \ p$. La chiave pubblica è $k_{pub} = \langle p, g, y\rangle$ e la chiave privata è $k_{priv} = \langle x\rangle$
- **Cifratura**
  il messaggio m è codificato come una sequenza binaria, trattata come un numero intero con $m < p$. Se $m \geq p$ si utilizza la cifratura a blocchi di $log_2 p$ bit ciascuno. Si sceglie a caso un intero $r \in [2, p-2]$ e si calcola la coppia di crittogrammi $c = g^r \ mod \ p$ e $d = m \cdot y^r \ mod \ p$

- **Decifrazione**
  si calcola $m = d \cdot c^{-x} \; mod \, p$

La difficoltà di calcolare il logaritmo discreto rende la chiave privata x sicura, anche se y e g sono pubblici. Siccome r è un intero casuale allora anche $y^r \; mod \, p$ è un intero casuale e $d = m \cdot y^r \; mod \, p$ è casuale e non fornisce alcuna informazione su m al crittoanalista. Per "estrarre" r da c occorre risolvere il problema del logaritmo discreto: se r è noto si può decifrare: $m = d \cdot y^{-r} \; mod \, p$, quindi occorre un nuovo numero casuale r per ogni nuovo messaggio.

7.9.1 Attacco

Si suppone che Alice invia a B i messaggi m_1, m_2 usando lo stesso r, quindi i crittogrammi sono $\langle c = g^r \; mod \, p, d_1 = y^r \cdot m_1 \; mod \, p \rangle$ e $\langle c = g^r \; mod \, p, d_2 = y^r \cdot m_2 \; mod \, p \rangle$. Si suppone che Eve viene a conoscenza di m_1 e dunque conosce $p, g, y, c, d_1, d_2, m_1$ e può trovare m_2 in tempo polinomiale

Chapter 8

Crittografia su curve ellittiche

8.1 Introduzione

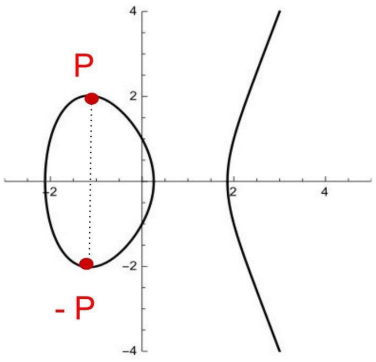
Negli ultimi vent'anni si è reso disponibile un sistema alternativo di crittografia a chiave pubblica, la **crittografia su curve ellittiche**, in grado di offrire prestazioni migliori e maggiore sicurezza rispetto ai sistemi a chiave pubblica di "prima generazione" (RSA, DH, ElGamal). In pratica a parità di sicurezza la crittografia su curve ellittiche richiede chiavi di dimensioni di gran lunga inferiori riducendo così il carico computazionale. L'idea è di sostituire negli algoritmi già esistenti le operazioni basate sull'algebra modulare con operazioni definite sui punti di una curva ellittica.

8.2 Curve ellittiche sui numeri naturali

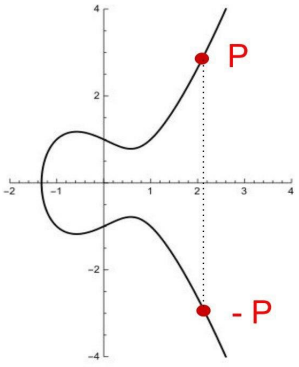
Una curva ellittica è un insieme di punti nel piano, si assume per il momento in $\mathbb{R}$, dove x e y soddisfano questa equazione:

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

$E(a,b)$ contiene il punto all'infinito O in direzione dell'asse y. Si assume inoltre che $4a^3+27b^2 \neq 0$, questa condizione assicura che il polinomio cubico $x^3 + ax + b$ non abbia radici multiple e di conseguenza la curva sia priva di punti singolari come "cuspidi" o "nodi" dove non sarebbe definita in modo univoco la tangente. Il grafico delle curve ellittiche sui reali può assumere una tra due possibili forme: una forma a due componenti (figura a sinistra) che si presenta quando il polinomio in x ha tre radici reali, e una forma con una sola componente (figura a destra) che si presenta quando il polinomio ha una sola radice reale. In entrambi i casi, le curve ellittiche presentano una simmetria orizzontale: ogni punto $P = (x,y)$ sulla curva si riflette rispetto all'asse delle ascisse nel punto $(x,-y)$ anch'esso sulla curva, che viene indicato con $-P$. L'immagine speculare rispetto all'asse x del punto all'infinito O è lo stesso O

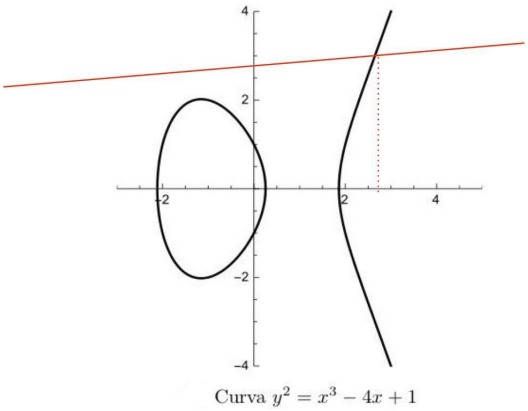


Curva $y^2 = x^3 - 4x + 1$

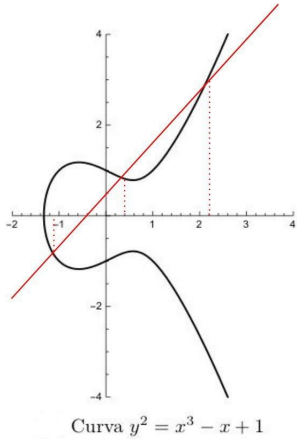


Curva $y^2 = x^3 - x + 1$

Proprietà: ogni retta interseca una curva ellittica in al più tre punti. Infatti intersecando una curva di terzo grado con una di primo grado e sostituendo nell'equazione della curva $y^2 = x^3 + ax + b$ l'espressione di y della retta si ottiene un'equazione di terzo grado in x che ha tre soluzioni reali o complesse, corrispondenti alle ascisse dei punti di intersezione tra la curva ellittica e la retta.



una soluzione reale e due soluzioni complesse e coniugate, quindi un punto reale di intersezione

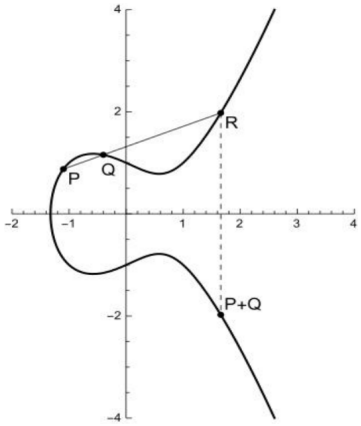


tre soluzioni reali, quindi tre punti di intersezione

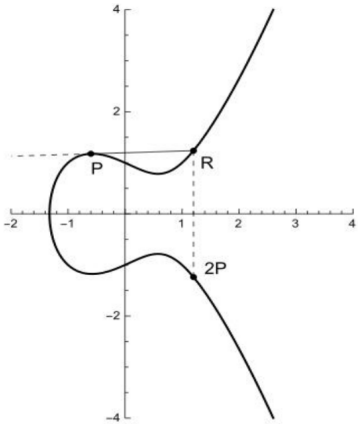
Quindi se una retta interseca la curva $E(a,b)$ in due punti P e Q, coincidenti se la retta è una tangente, allora la retta interseca $E(a,b)$ anche in un terzo punto R. Dati tre punti $P,Q,R \in E(a,b)$, se P,Q e R sono disposti su una retta, si pone $P+Q+R=O$. Da questa definizione si ricava la regola per sommare due punti P e Q

- si considera la retta passante per P e Q, oppure l'unica tangente alla curva in P nel caso P e Q coincidano
- si determina il punto di intersezione R tra la curva e la retta per P e Q, oppure tra la curva e la tangente in P per $P=Q$
- si definisce somma di P e Q il punto simmetrico a R rispetto all'asse delle ascisse, ovvero si pone $P+Q=-R$

La somma è ben definita in quanto anche $-R$ è un punto della curva



somma di due punti P e Q



raddoppio di un punto P

La formulazione algebrica dell'operazione di addizione permette di calcolare le coordinate del punto $P+Q$ a partire dalle coordinate di P e Q. Siano $P=(x_P,y_P)$ e $Q=(x_Q,y_Q)$ due punti distinti della curva:

1. se $P \neq \pm Q$, la somma $P+Q$ è data dal punto S (ovvero $-R$) di coordinate $x_S = \lambda^2 - x_P - x_Q$ e $y_S = -y_P + \lambda(x_P - x_S)$ dove $\lambda = (y_Q - y_P)/(x_Q - x_P)$ è il coefficiente angolare della retta passante per P e Q
2. se $P=Q$, sommare P e Q corrisponde a raddoppiare P. Se $y_P \neq 0$, le coordinate del punto $S = P+P = 2P$ sono ancora date dalle formule del punto precedente, dove ora $\lambda = (3x_p^2 + \alpha)/2y_P$. Se invece $y_P = 0$, si pone $2P = 0$
3. se $Q=-P$ allora $S = P+Q = P+(-P) = 0$

La somma soddisfa le 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**, $\forall P \in E(a,b)$ esiste un unico $Q = -P \in E(a,b)$ tale che $P + Q = Q + P = O$
- **associatività**, $\forall P, Q, E \in E(a,b), P + (Q + R) = (P + Q) + R$
- **commutatività**, $\forall P, Q \in E(a,b), P + Q = Q + P$

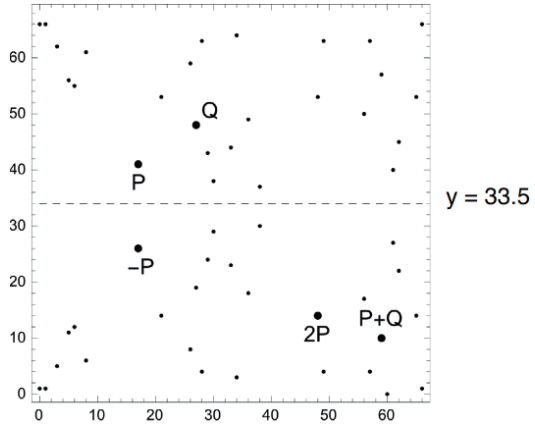
8.3 Curve ellittiche su campi finiti

Gli algoritmi crittografici hanno infatti bisogno di un'aritmetica veloce e precisa, e pertanto non possono utilizzare le curve ellittiche sui reali che richiedono elaborazioni lente e inaccurate a causa degli errori di arrotondamento. Esistono due famiglie principali:

- **Curve ellittiche prime**, variabili e coefficienti ristretti agli elementi del campo $\mathcal{Z}_p$ con p numero primo. Si impone $p > 3$ in modo da poter ridurre l'equazione generale di una curva ellittica a $y^2 = x^3 + ax + b$. Presi dunque $a, b \in \mathcal{Z}_p$, la curva ellittica prima $E_p(a,b)$ è definita come l'insieme dei punti che soddisfano l'equazione $y^2 = x^3 + ax + b$ modulo p insieme al punto all'infinito

  $$E_p(a,b) = \{(x,y) \in \mathcal{Z}_p^2 \mid y^2 \ mod \ p = (x^3 + ax + b) \ mod \ p\} \cup \{O\}$$

  Una curva ellittica prima contiene un numero finito di punti, e non è più rappresentata da una curva nel piano bensì da una nuvola di punti come mostrato in figura per $p = 67, a = -1, b = 1$. La curva ellittica prima $E_p(a,b)$ risulta simmetrica rispetto alla retta $y = p/2$. Il punto $(x, -y \ mod \ p) = (x, p - y)$ definisce l'inverso di P e si indica con $-P$. Per esempio il punto $P = (17, 41)$ appartiene alla curva $E_{67}(-1,1)$, quindi anche il punto $-P = (17, 67 - 41) = (17, 26)$ le appartiene

- **Curve ellittiche binarie**, variabili e coefficienti assumono valore nel campo $GF(2^m)$. L'equazione che definisce la curve ellittica assume una forma leggermente diversa:

  $$y^2 + xy = x^3 + ax^2 + b$$

  dunque anche le formule algebriche della somma di punti sono diverse

Esempio. Si disegna la curva $E_5(4,4)$, quindi si lavora modulo 5, allora si verifica che $4a^3 + 27b^2 \ mod \ 5 \neq 0 \Rightarrow 4 \cdot 4^3 + 27 \cdot 4^2 \ mod \ 5 \neq 0 \Rightarrow 3 \ mod \ 5 \neq 0$. Per trovare l'**ordine** di una curva, ovvero il suo numero di punti, si considera l'equazione $y^2 = x^3 + ax + b \Rightarrow y^2 = x^3 + 4x + 4 \ mod \ 5$ e si trovano i **residui quadratici** in $\mathcal{Z}_5$, cioè gli elementi del campo che ammettono radice quadrata in $\mathcal{Z}_5$

y	0	1	2	3	4
y^2	0	**1**	**4**	4	1

quindi i punti di $\mathcal{Z}_5$ che hanno una radice nel campo sono 1 e 4, adesso si valutano i punti:

x	y^2	y	
0	4	$2, -2$	$\Rightarrow (0,2), (0,3)$
1	4	$2, -2$	$\Rightarrow (1,2), (1,3)$
2	0	0	$\Rightarrow (2,0)$
3	3	*non ci sono punti della curva di ascisse* 3	
4	4	$2, -2$	$\Rightarrow (4,2), (4,3)$

L'ordine della curva sono 8 compreso O.

Il teorema di **Hasse** annuncia che l'ordine N di una curva ellittica $E_p(a,b)$ verifica la disuguaglianza $|N-(p+1)| \leq 2\sqrt{p}$

8.4 Funzione one-way trap door

Per le curve ellittiche si può definire una funzione one-way trap door analoga al logaritmo discreto nell'algebra modulare. L'addizione di punti di una curva ellittica presenta delle analogie con l'operazione di prodotto modulare. In particolare, fissato un intero positivo k si può mettere in relazione l'elevamento alla potenza k di un intero in modulo con la **moltiplicazione scalare** per k di un punto P di una curva ellittica, operazione che consiste nel sommare P con sé stesso k volte. Entrambe le operazioni si possono eseguire in tempo polinomiale. **Algoritmo dei raddoppi ripetuti**:

1. si scrive k come somma di potenze del due, $k = \sum_{i=0}^{t} k_i 2^i$ con $t = \lfloor log_2 k \rfloor$
2. si calcolano in successione i punti $2P, 4P, ..., 2^t P$, ottenendo ciascuno dal raddoppio del precedente
3. infine si calcola Q come somma dei punti $2^i P$ per $0 \leq i \leq t$ tale che $k_i = 1$

Si immagina di voler calcolare $13P$, che viene riscritto come $(8+4+1)P = 8P+4P+P$. A questo punto si raddoppia:

$$\begin{array}{c} \boldsymbol{P} \\ \downarrow \quad \textit{1 raddoppio} \\ 2P \\ \downarrow \quad \textit{1 raddoppio} \\ 2(2P) = \boldsymbol{4P} \\ \downarrow \quad \textit{1 raddoppio} \\ 2(4P) = \boldsymbol{8P} \end{array}$$

Adesso si hanno tutti i termini che servono (indicati in grassetto), quindi per calcolare $8P + 4P + P$ occorrono 3 raddoppi e 2 somme di punti. In generale servono $\lfloor log_2 k \rfloor$ raddoppi e $O(log k)$ somme. Si considera ora l'operazione inversa della moltiplicazione scalare su una curva ellittica, ovvero dati due punti P e Q trovare, se esiste, il più piccolo intero k tale che $Q = kP$. Questo problema è "difficile" ed è noto come il **problema del logaritmo discreto per le curve ellittiche**. 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, chiaramente improponibile se k è sufficientemente grande.

8.5 Protocollo DH su curve ellittiche

È una riproposizione basato sul problema del logaritmo discreto sostituendo il prodotto con la somma di punti e l'esponenziazione con la moltiplicazione scalare di un intero per un punto della curva. Operazioni:

1. Alice e Bob scelgono pubblicamente una curva ellittica prima $E_p(a,b)$ e un punto $B \in E_p(a,b)$ di ordine elevato, dove l'**ordine di un punto** è il più piccolo intero n tale che $nB = O$
2. Alice sceglie un intero positivo casuale $n_A < n$ come propria chiave privata, genera una chiave pubblica $P_A = n_A B$ e la spedisce in chiaro a Bob. La chiave pubblica corrisponde dunque ad un punto della curva scelto casualmente
3. Analogamente Bob estrae un intero positivo casuale $n_B < n$ come propria chiave privata, genera una chiave pubblica $P_B = n_B B$ e la spedisce in chiaro a Alice. Di nuovo, la chiave pubblica è un punto della curva scelto casualmente
4. Alice riceve P_B e calcola $n_A P_B = n_A n_B B = S$ usando la sua chiave privata n_A
5. Bob riceve P_A e calcola $n_B P_A = n_B n_A B = S$ usando la sua chiave privata n_B
6. A questo punto Alice e Bob condividono lo stesso punto S determinato dalle scelte casuali di entrambi. Per trasformare S in una chiave segreta k per la cifratura simmetrica convenzionale occorre convertirlo in un numero intero. Ad esempio si pone $k = x_S \ mod \ 2^{256}$

Attacco passivo: Eve conosce la curva $E_p(a,b), B, P_A$ e P_B, per calcolare il punto k ha bisogno di n_A o n_B ovvero deve risolvere il problema del logaritmo discreto su curve ellittiche. Il protocollo quindi resiste agli attacchi passivi, ma è comunque vulnerabile agli attacchi attivi di tipo man-in-the-middle esattamente come il protocollo DH

8.6 Protocollo di ElGamal su curve ellittiche

Si presenta ora un algoritmo per lo **scambio di messaggi cifrati**, la cui versione classica è nota come cifrario a chiave pubblica di ElGamal. Il protocollo richiede di incapsulare il messaggio m in un punto di una curva ellittica. Non è un compito semplice, infatti usando m come ascissa, la probabilità di trovare un punto della curva è pari alla probabilità che $m^3 + am + b \; mod \; p$ sia un residuo quadratico, che è circa $\simeq 1/2$. Esistono tuttavia degli algoritmi randomizzati molto efficienti, tra questi vi è l'**algoritmo di Koblitz**: si sceglie un intero h tale che $(m+1)h < p$ e si considerano come potenziali valori dell'ascisse del punto della curva gli interi $x = mh + i$, al variare di i da 0 a $h-1$.

```
// a, b, p parametri curva
KOBLITZ(m, h, a, b, p){

    for(i = 0; i < h; i++){
        x = mh + i;
        z = (x^3 + ax + b) mod p;
        if(z residuo quadratico) {
            y = √z;   // costo polinomiale
            return Pm = (x, y);
        }
    }
    return "failure"; // l'algoritmo ha fallito
}
```

La probabilità di fallimento è $\simeq (1/2)^h$, quindi la probabilità di successo è $\simeq 1 - (1/2)^h$. Per risalire al messaggio dal punto $Pm = (x, y)$ individuato basta calcolare $m = \lfloor \frac{x}{h} \rfloor = \lfloor \frac{mh+i}{h} \rfloor = \lfloor m + \frac{i}{h} \rfloor = m$ perché $\frac{i}{h} < 1$. La fase di preparazione è finita, ora si passa di cifratura e decifrazione. Anche in questo caso occorre fissare una curva $E_p(a, b)$ e un punto base B di ordine n elevato tale che $B \in E_p(a, b)$. Ogni utente U costruisce la propria coppia $k[pub], k[priv]$ scegliendo un intero casuale $n_U < n$ come chiave privata e pubblicando la chiave pubblica $P_U = n_U B$. Si suppone ora che Alice vuole inviare un messaggio m a Bob

Cifratura. Alice incapsula il messaggio m in un punto con l'algoritmo di Koblitz, e poi prepara il crittogramma: sceglie un intero $r < n$ casuale e calcola due punti della curva $V = rB, W = Pm + rP_{Bob}$, entrambi in tempo polinomiale con l'algoritmo dei raddoppi ripetuti. P_{Bob} è la chiave pubblica di Bob, quindi invia a Bob la coppia di punti $\langle V, W \rangle$

Decifrazione. Bob riceve la coppia di punti $\langle V, W \rangle$ e decifra calcolando $W - n_{Bob}V = Pm + rP_{Bob} - n_{Bob}(rB) = Pm + r(n_{Bob}B) - n_{Bob}(rB) = Pm$, poichè $P_{Bob} = n_{Bob}B$. Quindi trasforma Pm nel messaggio m

Il primo *attacco passivo* che il crittoanalista può cercare di fare è trovare n_{Bob} da P_{Bob} ma $P_{Bob} = n_{Bob}B$ che è il logaritmo discreto su curve ellittiche: problema "difficile". Un'altra possibilità è trovare r perché conoscendo r può decifrare $W = Pm + rP_{Bob} \Rightarrow Pm = W - rP_{Bob}$

8.7 Sicurezza della crittografia su curve ellittiche

La sicurezza della crittografia su curve ellittiche è strettamente legata alla difficoltà di calcolare il logaritmo discreto di un punto, problema per cui non è noto alcun algoritmo efficiente di risoluzione. Nonostante manchi una dimostrazione formale della sua intrattabilità, questo problema è considerato estremamente difficile, e in particolare molto più difficile dei tradizionali problemi della fattorizzazione degli interi e del logaritmo discreto nell'algebra modulare. Infatti per questi problemi esiste un algoritmo subesponenziale chiamato **index calculus** per calcolare il logaritmo discreto, che può essere utilizzato per attaccare sia il protocollo DH che il cifrario RSA. L'algoritmo index calculus frutta una struttura algebrica dei campi finiti che non è presente sulle curve ellittiche. Ad oggi nessuno è stato capace di progettare algoritmi di tipo index calcus per il problema del logaritmo discreto per le curve ellittiche: quindi i protocolli per tali curve sembrano invulnerabili a questo tipo di attacchi. Al momenti il migliore attacco noto al problema del logaritmo discreto per le curve ellittiche, detto **Pollard ρ**, richiede in media $O(2^{b/2})$ operazioni per chiavi di b bit ed è dunque pienamente esponenziale. Esistono attacchi particolarmente efficaci contro alcune famiglie di curve, ma non sono utili in generale poiché queste famiglie sono note e facilmente evitate.

Chapter 9

La firma digitale

9.1 Introduzione

In origine i metodi crittografici sono stati sviluppati per garantire la confidenzialità delle comunicazioni, in particolare tra coppie di persone in ambienti ristretti. Con la diffusione delle reti, in particolare di Internet, il problema si è però enormemente esteso e ha assunto connotazioni nuove. Sono così emerse tre funzionalità importanti che sono oggi richieste ai protocolli crittografici:

- **Identificazione**, un sistema di elaborazione, isolato o in rete, deve essere in grado in accertare l'identità di un utente che richiede di accedere ai suoi servizi
- **Autenticazione**, il destinatario di un messaggio deve essere in grado di accertare l'identità del mittente e **l'integrità** del crittogramma ricevuto, cioè che non sia stato modificato o sostituito nella trasmissione
- **Firma digitale**, è la funzionalità più complessa e risulta necessaria se il mittente e il destinatario non sono tenuti a fidarsi l'uno dell'altro. La firma digitale deve possedere tre requisiti:
  1. Il mittente non deve poter negare di aver inviato un messaggio m
  2. Il destinatario deve essere in grado di autenticare il messaggio
  3. Il destinatario non deve poter sostenere che un messaggio $m' \neq m$ è quello inviatogli dal mittente

  Questi requisiti devono essere verificabili da una terza parte, cioè un "giudice" possibilmente chiamato a certificare il corretto svolgimento della comunicazione.

Queste tre funzionalità non sono indipendenti, ma ciascuna estende le precedenti poiché l'autenticazione di un messaggio garantisce l'identificazione del mittente, e l'apposizione della firma garantisce l'autenticazione del messaggio. Ogni funzionalità è utilizzata per contrastare gli attacchi attivi (in particolare di tipo man-in-the-middle). Esistono varie realizzazione algoritmiche basate sui cifrari asimmetrici e simmetrici. A questo scopo si introduce una nuova classe di funzioni: **funzioni hash one-way**

9.2 Funzioni hash one-way

Una funzione hash $f : X \to Y$ è una funzione tale che $n = |X| >> m = |Y|$, ovvero è il dominio è molto più numeroso del codominio e ci saranno più elementi del dominio che hanno la stessa immagine hash (*collisioni*). Una buona funzione hash deve assicurare che:

1. i sottoinsiemi $X_1, ..., X_m$ abbiano circa la stessa cardinalità in modo che due elementi x_h, x_k estratti a caso da X abbiano probabilità circa pari a $1/m$ di avere la stessa immagine in Y
2. elementi di X molto "simili" appartengano a due sottoinsiemi diversi. Per esempio se X è un insieme di interi, due elementi con valori prossimi devono avere immagini diverse
3. gestioni 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

Quando le funzioni hash sono applicate in crittografia le proprietà più interessanti sono però un po' diverse, in particolare sono importanti le funzioni hash one-way tali che:

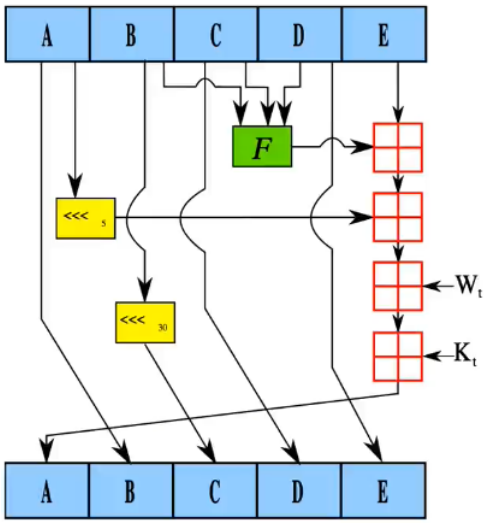
1. per ogni $x \in X$ è computazionalmente facile calcolare $y = f(x)$
2. **proprietà one-way**, per la maggior parte degli elementi $y \in Y$ è computazionalmente difficile determinare un x tale che $f(x) = y$
3. **proprietà claw-free**, è computazionalmente difficile determinare una coppia di elementi x_1, x_2 in X tali che $f(x_1) = f(x_2)$

Le funzioni hash one-way più usate in crittografia sono:

- **MD5**, riceve in input una sequenza S di 512 bit e produce un'immagine di 128 bit. La sequenza è digerita riducendone la lunghezza ad un quarto. È stato proposto da Rivest nel 1992 e fa parte di una famiglia di funzioni. MD5 non resiste alle collisioni, e nel 2004 sono state individuate delle debolezze serie. Oggi la sua sicurezza si considera severamente compromessa.
- **RIPEMD-160**, produce immagini di 160 bit ed è esente dai difetti di MD5. Nata nel 1995 nell'ambito di un progetto dell'Unione Europea, è la versione matura delle funzioni MD
- **SHA**, opera su sequenze lunghe fino a 2^{64} bit e produce immagini di 160 bit. Progettata da NIST e NSA è una funzione crittograficamente sicura che soddisfa i requisiti delle funzioni hash one-way, e genera immagini molto diverse per sequenze molto simili. Fa parte di una famiglia di funzioni Hash nate dal 1993 progettate da NIST e NSA. Si adotta quando la proprietà di claw-free è cruciale per la sicurezza del sistema

9.2.1 SHA-1

SHA-1 è molto usata nei protocolli crittografici anche se non è più certificata come standard. Tutte le altre funzioni hanno una struttura molto simile a SHA-1. Opera su blocchi di 160 bit contenuti in un buffer di 5 registri A, B, C, D, E di 32 bit ciascuno, in cui sono caricati inizialmente dei valori fissi e pubblici. Il messaggio m di cui si vuole calcolare l'immagine è concatenato con una sequenza di *padding* che ne rende la lunghezza multipla di 512 bit. La struttura di massima per il calcolo della funzione è indicata dalla seguente figura.



A, B, C, D, E sono registri di 32 bit
F è una funzione non lineare
$<<<_n$ denota una rotazione del bit di sinistra di n posti
n varia per ogni operazione
$\boxplus$ denota l'addizione modulo 2^{32}
K_t è una costante
W_t blocco di 32 bit ottenuto tagliando e rimescolando i blocchi del messaggio

Il contenuto dei registri varia nel corso dei cicli (all'inizio sono caricati valori fissi e pubblici) in cui questi valori si combinano tra loro e con blocchi di 32 bit provenienti dal messaggio W, nonché con alcuni parametri relativi al ciclo. Alla fine del procedimento, i registri contengono SHA-1(m)

9.3 Identificazione

Per introdurre il problema dell'identificazione si considera l'accesso di un utente alla propria casella di posta elettronica, o ai file personali memorizzati su un calcolatore ad accesso riservato ai membri della sua organizzazione. L'utente inizia il collegamento inviando in chiaro *login* e *password*. Se il canale è protetto in lettura e scrittura, un attacco può essere sferrato solo da un utente locale al sistema, per esempio l'*amministratore* che ha accesso a tutti i file memorizzati; o da un hacker che è riuscito a intrufolarsi nel sistema. Ne segue che il meccanismo di identificazione deve prevedere una opportuna cifratura delle password, che può essere realizzata in modo semplice ed efficace impiegando funzioni hash one-way. Il metodo che ora si descrive è utilizzato per esempio nei sistemi UNIX: quando un utente U fornisce per la prima volta una password P, il sistema associa a U due sequenze binarie S, Q che memorizza nel file delle password al posto di P. Il seme S è prodotto da un generatore pseudo-casuale; la $Q = h(PS)$ è l'immagine della sequenza ottenuta concatenando S alla password di U. Ad ogni successiva connessione dell'utente il sistema:

- recupera il seme S dal file delle password;
- concatena S con la password fornita da U
- calcola l'immagine one-way della nuova sequenza: $h(PS)$
- se $h(PS) = Q$ l'identificazione ha successo

In questo modo un eventuale accesso illecito al file delle password non fornisce alcuna informazione interessante, poiché risulta computazionalmente difficile ricavare la password originale della sua immagine one-way. Se il canale è insicuro, la password può essere intercettata durante la sua trasmissione in chiaro: è quindi opportuno che il sistemi non maneggi mai direttamente la password ma una sua immagine inattaccabile

9.3.1 Protocollo di identificazione basato sull'RSA

Siano $\langle e, n\rangle$ e $\langle d\rangle$ le chiavi pubbliche e private di un utente U che vuole accedere ai servizi offerti da un sistema S

1. In corrispondenza a una richiesta di accesso proveniente da U, il sistema S genera un numero casuale $r < n$ e lo invia in chiaro a U
2. U "decifra" il messaggio r calcolando $f = r^d \ mod \ n$ con la sua chiave privata $\langle d\rangle$ e spedisce a S il valore di f. Questo valore prende il nome di **firma** di U su r
3. S verifica la correttezza del valore ricevuto "cifrandolo" con la chiave pubblica $\langle e, n\rangle$ di U come $f^e \ mod \ n$ e controllando che il risultato coincida con r,

Si noti l'inversione d'ordine delle operazioni di cifratura e decifrazione rispetto all'impiego standard dell'RSA: ciò è possibile perché le due funzioni sono commutative, ovvero $(x^e \ mod \ n)^d \ mod \ n = (x^d \ mod \ n)^e \ mod \ n$. Il valore f può essere generato solo da U che possiede la chiave privata $\langle d\rangle$: quindi se il passo 3. va a buon fine il sistema ha la garanzia che l'utente che ha richiesto l'identificazione sia effettivamente U, anche se il canale è insicuro. Questo protocollo funziona bene se il sistema è onesto. Un sistema disonesto potrebbe inviare una r non casuale, scelta di proposito per ricavare qualche informazione sulla chiave privata di U. Si vedrà in seguito un protocollo alternativo a "conoscenza zero" che impedisce che da una comunicazione si possa estrarre più di quanto sia nelle intenzioni del comunicatore.

9.4 Autenticazione

Il processo di identificazione è volto a stabilire l'identità di un utente ma non si occupa dei suoi messaggi. Si considera il caso in cui Bob deve autenticare il messaggio accertando l'identità di Alice e l'integrità del messaggio m. Alice e Bob concordano una chiave segreta k.

1. Alice allega al messaggio un **MAC**, $A(m, k)$, allo scopo di garantire la provenienza e l'integrità del messaggio, e spedisce se non è richiesta confidenzialità la coppia $\langle m, A(m, k)\rangle$ in chiaro, oppure, cifra m e spedisce $\langle \mathcal{C}(m, k'), A(m, k)\rangle$, dove $\mathcal{C}$ è la funzione di cifratura e k' è la chiave pubblica o segreta del cifrario scelto
2. Bob entra in possesso di m (dopo averlo eventualmente decifrato) ed essendo a conoscenza di A e k può ricalcolare $A(m, k)$, confrontare il risultato con il valore inviatogli da Alice e verificare che il MAC ricevuto sia proprio quello corrispondente al messaggio a cui risulta allegato. Se la verifica si conclude con successo il messaggio è autenticato, altrimenti Bob scarta il messaggio.

MAC è un'immagine breve del messaggio, che può essere stata generata solo da un mittente conosciuto dal destinatario, previ opportuni accordi. Ne sono state proposte varie realizzazioni basate su cifrari asimmetrici, simmetrici e sulle funzioni hash one-way. Quest'ultimo approccio conduce a protocolli molto economici che sono attualmente i più usati in pratica. Si definisce il MAC sulla concatenazione delle sequenza m e k come $A(m, k) = h(mk)$. Risulta computazionalmente difficile per un crittoanalista scoprire la chiave segreta k. Infatti, sebbene la funzione h sia nota a tutti e m possa viaggiare in chiaro o essere scoperto per altra via, k viaggia all'interno del MAC: per recuperare k si dovrebbe quindi invertire la h. Da ciò consegue che il crittoanalista non può sostituire facilmente il messaggio m con un altro messaggio m' perché dovrebbe allegare alla comunicazione m' l'informazione $A(m', k)$ che può produrre solo conoscendo la chiave segreta k. L'impiego di un qualsiasi cifrario a blocchi in modalità CBC consente di generare un MAC: basta prendere il blocco finale del crittogramma che è funzione dell'intero messaggio

9.5 Firma digitale

La *firma manuale*, utilizzata comunemente per provare l'autenticità o la paternità di un documento soddisfa i seguenti requisiti fondamentali:

1. La firma è **autentica** e **non falsificabile**, dunque costituisce prova che chi l'ha prodotta è veramente colui che ha sottoscritto il documento
2. La firma **non è riutilizzabile**, e quindi risulta legata strettamente al documento su cui è stata apposta
3. Il documento firmato **non è alterabile**, e quindi chi ha prodotto la firma è sicuro che questo si riferirà solo al documento sottoscritto nella sua forma originale
4. La firma **non può essere ripudiata** da chi l'ha apposta, e costituisce dunque prova legale di un accordo o di una dichiarazione contenuta nel documento

La *firma digitale* non può semplicemente consistere di una digitalizzazione del documento originale firmato manualmente, poiché un crittoanalista potrebbe "tagliare" dal documento digitale la parte contenente la firma e "copiarla" su un altro documento. Quindi, a differenza della firma manuale, la firma digitale deve avere una forma che dipende dal documento su cui viene apposta, per essere inscindibile da questo. Per progettare firme digitali si possono usare sia i cifrari simmetrici che quelli asimmetrici. Sia U un generico utente, $k_U[priv]$ e $k_U[pub]$ le sue chiavi, $\mathcal{C}$ e $\mathcal{D}$ le funzioni di cifratura e decifrazione del cifrario asimmetrico. Un primo protocollo è il seguente:

Protocollo 1. Messaggio m in chiaro e firmato

Firma: l'utente U genera la firma $f = \mathcal{D}(m, k_U[priv])$ per m e spedisce all'utente V la tripla $\langle U, m, f\rangle$

Verifica: l'utente V riceve la tripla $\langle U, m, f\rangle$ e verifica l'autenticità della firma f calcolando $\mathcal{C}(f, k_U[pub])$ e controllando che questo valore sia uguale a m. L'indicazione del mittente U consente a V di selezionare la chiave pubblica $k_U[pub]$ da utilizzare nel calcolo

I processi di firma e verifica impiegano le funzioni $\mathcal{C}$ e $\mathcal{D}$ del cifrario in ordine inverso a quello standard: questa particolarità impone che le due funzioni siano commutative ossia $\mathcal{C}(\mathcal{D}(m)) = \mathcal{D}(\mathcal{C}(m)) = m$. Il protocollo 1 soddisfa i requisiti della firma manuale:

1. la firma f è autentica e non falsificabile, poiché la chiave $k_U[priv]$ è nota solo a U e la funzione $\mathcal{D}$ è one-way
2. la firma f non è riutilizzabile su un altro documento $m' \neq m$ poiché essa è immagine di m
3. il documento firmato $\langle U, m, f\rangle$ non può essere alterato se non da U, pena la non consistenza tra m e f
4. solo U può aver prodotto f, quindi U non può ripudiare la firma

Il protocollo è definito per un particolare utente U ma non per un particolare destinatario. Chiunque può convincersi dell'autenticità della firma facendo uso solo della chiave pubblica di U. Il protocollo 1 deve essere visto solo come schema di principio. Infatti esso composta lo scambio di un messaggio di lunghezza doppia dell'originale poiché la dimensione della firma è paragonabile alla dimensione del messaggio. Inoltre il messaggio non può essere cifrato poiché è ricavabile pubblicamente dalla firma attraverso la verifica di questa

Protocollo 2. Messaggio m cifrato e firmato

Firma e cifratura: il mittente U genera la firma $f = \mathcal{D}(m, k_U[priv])$ per m, calcola il *crittogramma firmato* $c = \mathcal{C}(f, k_V[pub])$ con la chiave pubblica del destinatario, ovvero, si incapsula la firma nel documento cifrato, e spedisce la coppia $\langle U, c\rangle$ a V

Decifrazione e verifica: il destinatario V riceve la coppia $\langle U, c\rangle$, decifra il crittogramma $\mathcal{D}(c, k_V[priv]) = f$, quindi cifra tale valore con la chiave pubblica di U ottenendo $\mathcal{C}(\mathcal{D}(m, k_U[priv]), k_U[pub]) = m$. In tal modo V ricostruisce il messaggio m, e se è significativo attesta l'identità di U. Infatti un utente diverso da U ha probabilità praticamente nulla di generare un crittogramma che assuma qualsivoglia significato accettabile se "cifrato" con la chiave pubblica di U

Si descrive un algoritmo che segua il protocollo 2 usando le funzioni di cifratura e decifrazione dell'RSA, dove $\langle d_U\rangle$, $\langle e_U, n_U\rangle$ sono le chiavi di U e $\langle d_V\rangle, \langle e_V, n_V\rangle$ sono le chiavi di V:

- U genera la firma del messaggio m come $f = m^{d_U} \bmod n_U$, cifra f con la chiave pubblica di V calcolando $c = f^{e_V} \bmod n_V$ e spedisce a V la coppia $\langle U, c\rangle$
- V riceve la coppia $\langle U, c\rangle$ e decifra c come $c^{d_V} \bmod n_V = f$, poi "decifra" f con la chiave pubblica di U calcolando $f^{e_U} \bmod n_U = m$. Se m è significativo, conclude che è autentico

Per la correttezza del procedimento è necessario che $n_U \leq n_V$ perché risulti $f < n_V$ e f possa essere cifrata correttamente e spedita a V. Ma ciò impedirebbe a V di inviare messaggi firmati e cifrati a U. Il sistema è allora organizzato come segue: si fissa pubblicamente un parametro H molto grande, per esempio $H = 2^{1024}$ e si fa in modo che le chiavi di firma siano minori di H, mentre, le chiavi di cifratura siano maggiori di H. Il valore di H assicura che tutte le chiavi possano essere scelte sufficientemente grandi, e quindi inattaccabili in modo esauriente. Il meccanismo di firma si presta tuttavia a diversi attacchi basati sulla possibilità che un crittoanalista riesca a procurarsi la firma di un utente su messaggi apparentemente privi di senso. Si suppone che utente U invii una risposta automatica (*ack*) al mittente ogni volta che riceve un messaggio m, dove i segnale di ack è il crittogramma della firma di U su m. Un crittoanalista attivo X può decifrare i messaggi inviati a U, infatti:

1. V invia il crittogramma $c = \mathcal{C}(f, k_U[pub])$ a U firmato $f = \mathcal{D}(m, k_V[priv])$
2. X intercetta c firmato inviato da V a U, lo rimuove dal canale e lo rispedisce a U, facendogli credere che c sia stato inviato da lui.
3. U decifra c ottenendo $f = \mathcal{D}(c, k_U[priv])$ e verifica la firma con la chiave pubblica di X ottenendo un messaggio $m' = \mathcal{C}(f, k_X[pub])$. Però $m' \neq m$, quindi è un messaggio privo di senso, ma il sistema manda l'ack c' a X sottoforma di crittogramma firmato $c' = \mathcal{C}(f', k_X[pub])$ dove $f' = \mathcal{D}(m', k_U[priv])$ è la firma di U su m'
4. Ora X può risalire da c' al messaggio originale m con le chiavi pubbliche di V e U attraverso la catena di calcoli: $\mathcal{D}(c', k_X[priv]) = f', \mathcal{C}(f', k_U[pub]) = m', \mathcal{D}(m', k_X[priv]) = f, \mathcal{C}(f, k_V[pub]) = m$

Per evitare questo attacco occorre che U blocchi l'ack automatico e invii l'ack solo dopo un esame preventivo di m e scartando i messaggi che non si desidera firmare. Una mossa per contrastare l'attacco precedente e molti altri attacchi di tipo algebrico consiste nell'evitare la *firma diretta* del messaggio m, ma apporre la firma digitale su un'immagine di m ottenuta mediante una funzione h hash one-way pubblica.

Protocollo 3. Messaggio m cifrato e firmato in hash

> **Firma e cifratura**: il mittente U calcola $h(m)$ e genera la firma $f = \mathcal{D}(h(m)), k_U[priv])$, calcola separatamente il crittogramma $c = \mathcal{C}(m, k_V[pub])$ e spedisce a V al tripla $\langle U, c, f \rangle$
>
> **Decifrazione e verifica**: il destinatario V riceve la tripla $\langle U, c, f \rangle$, decifra c come $m = \mathcal{D}(c, k_V[priv])$ ottenendo m, calcola separatamente $h(m)$ e $\mathcal{C}(f, k_U[pub])$: se questi due valori sono uguali conclude che il messaggio è identico

Il protocollo 3 richiede lo scambio di una quantità maggiore di dati rispetto al protocollo 2 a firma diretta, ma l'incremento è trascurabile, la firma può essere calcolata velocemente.

9.6 Certification Authority

Un algoritmo crittografico è tanto robusto quanto la sua sicurezza delle sue chiavi: le chiavi di cifratura sono pubbliche e quindi non richiedono un incontro diretto tra gli utenti per il loro scambio. Con un attacco di tipo man-in-the-middle, un crittoanalista attivo può intromettersi proprio in questa fase iniziale del protocollo, pregiudicando il suo corretto svolgimento. Per evitare questi attacchi sono nate le **Certification Authority** sono enti che garantiscono la validità delle chiavi pubbliche e ne regolano l'uso, gestendo la distribuzione sicura delle chiavi alle due entità che vogliono comunicare. La CA autentica l'associazione $\langle$ utente, chiave pubblica $\rangle$ emettendo un certificato digitale. Il certificato digitale consiste della chiave pubblica e di 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 a tutti e protetto da attacchi in scrittura non autorizzati. La chiave pubblica della CA è nota agli utenti che la mantengono protetta da attacchi esterni e la utilizzano per verificare la firma della CA stessa sui certificati. In dettaglio un certificato digitale contiene:

- una indicazione del suo formato (numero versione)
- il nome della CA che l'ha rilasciato
- un numero seriale che individua univocamente questo certificato all'interno della CA emittente
- la specifica dell'algoritmo utilizzato dalla CA per creare la firma elettronica, combinata con una descrizione del formato dei parametri di cui esso ha bisogno
- il periodo di validità del certificato (data inizio e data fine)

- il nome dell'utente a cui questo certificato si riferisce e una serie di informazioni a esso legate
- Un'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

Se un utente U vuole comunicare con un altro utente V può richiedere la chiave pubblica di quest'ultimo alla CA, che risponde inviando a U il certificato digitale di V. Poiché U conosce la chiave pubblica della CA, egli può controllare l'autenticità del certificato verificandone il periodo di validità e la firma prodotta dalla CA su di esso. Se tutti questi controlli vanno a buon fine, la chiave pubblica di V nel certificato è corretta e U la può utilizzare per avviare la comunicazione cifrata con V. Un crittoanalista potrebbe intromettersi soltanto falsificando quella certificazione, ma si assume che la CA sia fidata e il suo archivio delle chiavi sia inattaccabile. Attualmente esistono in ogni stato diverse CA organizzate ad albero: in questo caso la verifica di un certificato è leggermente 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 della chiave pubblica della CA. È desiderabile che il certificato di U sia reperibile pubblicamente in un sito contenente una lista di certificato prodotti da CA, ciò consente all'utente di interagire con la CA una volta soltanto. In questo modo la gestione delle chiavi diventa decentralizzata

Protocollo 4. Messaggio m cifrato, firmato in hash e certificato

Firma, cifratura e certificazione, il mittente U si procura il certificato $cert_V$ di V e verifica che sia autentico. Poi U calcola $h(m)$, genera la firma $f = \mathcal{D}(h(m), k_U[priv])$, calcola il crittogramma $c = \mathcal{C}(m, k_V[pub])$ e spedisce a V la tripla $\langle cert_U, c, f\rangle$. Si noti che $cert_U$ contiene la chiave $k_U[pub]$ e la specificazione della funzione h utilizzata

Verifica del certificato, il destinatario V riceve la tripla $\langle cert_U, c, f\rangle$ e verifica l'autenticità di $cert_U$ (e dunque $k_U[pub]$) utilizzando la propria copia $k_{CA}[pub]$

Decifrazione e verifica della firma, V decifra il crittogramma c mediante la sua chiave privata ottenendo $m = \mathcal{D}(c, k_V[priv])$, verifica l'autenticità della firma apposta da U su m controllando che risulti $C(f, k_U[pub]) = h(m)$

Un punto debole di questo meccanismo è rappresentato dai certificati revocati, cioè non più validi. La CA mettono a disposizione degli utenti un archivio pubblico contenente i certificati non più validi: la frequenza di controllo di questi certificati e le modalità della loro comunicazione agli utenti sono punti cruciali per la sicurezza e l'efficienza dei sistemi. Attacchi man-in-the-middle possono essere evitati se si stabilisce un incontro diretto tra utente e CA nel momento in cui si istanzia il sistema asimmetrico dell'utente

9.7 Protocollo zero-knowledge

Si studia lo schema matematico **zero-knowledge** nell'ambito dei protocolli di scambio d'informazione tra un **prover** $\boldsymbol{P}$ e un **verifier** $\boldsymbol{V}$. La caratteristica principale di uno scambio di messaggi zero-knowledge è che il prover P riuscirà a dimostrare al verifier V di essere in possesso di una facoltà particolare o di una conoscenza specifica su un argomento senza comunicargli alcun'altra informazione salvo l'evidenza del suo possesso di tale facoltà o conoscenza. Si introduce l'argomento con il seguente scenario:

Scenario. P sostiene di essere capace di capire se i granelli di sabbia in una scatola sono pari o dispari. V vuole controllare se l'affermazione di P è vera e P vuole convincere V di avere questa capacità senza svelargli il sistema che usa. P e V adottano un protocollo a conoscenza zero: V mette dei granelli di sabbia all'interno di una scatola, e la risposta di P è un bit, 0 nel caso in cui i granelli sono in numero pari, 1 nel caso in cui sono dispari. Si suppone che P risponda 0. Ora V esegue le seguenti operazioni iterandole per k volte se stabilirà che l'affermazione di P è vera, o arrestandosi prima se scoprirà che P è un impostore:

1. V chiede a P di voltarsi per non osservare quello che farà
2. V sceglie un bit random e lanciando una moneta
3. se $e = 0$ allora V toglie un granello di sabbia e lo intasca, altrimenti non fa nulla
4. V chiede a P di guardare di nuovo la spiaggia e di rispondere nuovamente se i granelli di sabbia sono pari o dispari
5. se le risposte di P ai passi i e $i-1$ sono consistenti tra loro V passa all'iterazione $i+1$, altrimenti, arresta il procedimento dichiarando che P è un impostore

La probabilità complessiva che P possa ingannare V è $1/2^k$. Alla fine V non ha imparato nulla del metodo di P, si convince solo di questa capacità di P se egli vince, per V, un numero sufficiente di sfide

I principi generali su cui si basano i protocollo zero-knowledge sono i seguenti:

- **Completezza**, se l'affermazione di P è vera, V ne accetta sempre la dimostrazione
- **Correttezza**, se l'affermazione di P è falsa (P disonesto), V può essere ingannato con probabilità $\leq 1/2^k$ per un valore arbitrario k scelto da lui stesso.
- **Conoscenza zero**, se l'affermazione di P è vera nessun verificatore V, anche se disonesto, può acquisire alcuna informazione su questo fatto salvo la sua veridicità

Si introduce ora il primo protocollo di identificazione zero-knowledge proposto da Fiat e Shamir. Questo protocollo è basato sulla difficoltà del calcolo della radice modulo un numero composto. *Fase di preparazione*: P sceglie due numeri primi molto grandi, calcola $n = pq$ e sceglie un intero positivo $s < n$. Calcola $t = s^2 \ mod \ n$, rende nota la coppia $\langle n, t \rangle$ e privata la terna $\langle p, q, s \rangle$. Il protocollo procede in questo modo: si itera per k volte (con k scelto da V):

1. V chiede a P di iniziare una iterazione
2. P genera un intero random $r < n$, calcola $u = r^2 \ mod \ n$ e comunica u a V. Si genera un nuovo valore di r e quindi di u a ogni iterazione: il valore di r può essere "bruciato" al passo 4
3. V genera un intero random $e \in \{0, 1\}$ e lo comunica a P
4. P calcola $z = rs^e \ mod \ n$ e invia z a V. Se $e = 0$ si ha $z = r$, se $e = 1$ si ha $z = rs \ mod \ n$
5. V calcola $x = z^2 \ mod \ n$. Se x è uguale a $ut^e \ mod \ n$ torna al passo 1, altrimenti blocca il protocollo senza identificare P

Se dopo k iterazioni, il protocollo non si è mai interrotto, allora P viene identificato. Segue le dimostrazioni di completezza e correttezza:

- **Completezza**, se $e = 0$ allora $x = ut^e \ mod \, n = u \ mod \, n$; se $e = 1$ allora $x = z^2 \ mod \, n = (rs^e)^2 \ mod \, n = ut \ mod \, n$. Quindi P supera tutte le sfide se conosce s
- **Correttezza**, se P è disonesto e conosce il valore di t che è pubblico ma non conosce il valore di s, può tentare di ingannare V prevedendo i valori di e che questi genererà al passo 3 del protocollo.

  Caso 1: P prevede di ricevere $e = 0$ allora si esegue il protocollo senza modificarlo, e se la previsione è corretta, P supera la sfida.

  Caso 2: P prevede di ricevere $e = 1$ allora P cambia il passo 2 del protocollo, ovvero, anziché mandare $u = r^2 \ mod \ n$, P invia $u = r^2 t^{-1} \ mod \ n$

  In entrambi i casi al passo 4 P invierà poi a V lo stesso valore $z = r \ mod \ n$. Al passo 5 se la previsione di P su e è corretta, P supera la sfida, infatti V scoprirà che $x = ut^e \ mod \ n = r^2 \ mod \ n$ e accetterà l'iterazione. Se la previsione è sbagliata la relazione non sarà verificata e sarà scoperto l'inganno. Quindi P disonesto riesce a ingannare V se è in grado di prevedere il bit e. Se i bit e sono generati casualmente, P inganna V con probabilità $1/2^k$

9.8 Protocollo SSL

Il **protocollo SSL** è alla base dei protocolli più diffusi nelle comunicazioni segrete. Garantisce **confidenzialità** e **affidabilità** delle comunicazioni su Internet, proteggendole da intrusioni, modifiche o falsificazioni. Permette la comunicazione tra computer che non conoscono le reciproche funzionalità. Si considera un utente U che desidera accedere tramite Internet a un servizio offerto dal sistema S. Il protocollo SSL garantisce la **confidenzialità** poiché la trasmissione è cifrata mediata un sistema ibrido, in cui un cifrario asimmetrico è utilizzato per costruire e scambiare le chiavi di sessione, e un cifrario simmetrico utilizza queste chiavi per criptare i dati nelle comunicazioni successive. Il protocollo SSL garantisce inoltre la **autenticazione** dei messaggi accertando l'identità dei due partner, attraverso un cifrario simmetrico o facendo uso di certificati digitali, e inserendo nei messaggi stessi un apposito MAC che utilizza una funzione hash one way crittograficamente sicura. SSL si innesta tra un protocollo di trasporto e affidabile (TCP/IP) e un protocollo di applicazione (e.g HTTP). Inoltre, SSL è completamente indipendente dal protocollo di applicazione sovrastante. SSL è organizzato su due livelli:

- **SSL record**, è al livello più basso, è connesso direttamente al protocollo di trasporto e ha l'obiettivo di incapsulare i dati spediti dai protocolli dei livelli superiori, assicurando *confidenzialità* e *integrità* della comunicazione
- **SSL handshake**, è al livello superiore e permette all'utente di sistema di autenticarsi, negoziare gli algoritmi di cifratura e firma e stabilire le chiavi per i singoli algoritmi crittografici e per il MAC. In sostanza SSL handshake crea un canale *sicuro*, *affidabile* e *autentico* tra utente e sistema, entro il quale *SSL record* fa viaggiare i messaggi

Una sessione di comunicazione SSL è innescata da uno scambio di messaggi preliminari (*handshake*) indispensabile per la creazione del canale sicuro. Attraverso questi messaggi il sistema S (server) e l'utente U (client) si identificano a vicenda e cooperano alla costruzione delle chiavi segrete da impiegare nelle comunicazioni simmetriche successive. I passi principali dell'SSL handshake sono i seguenti:

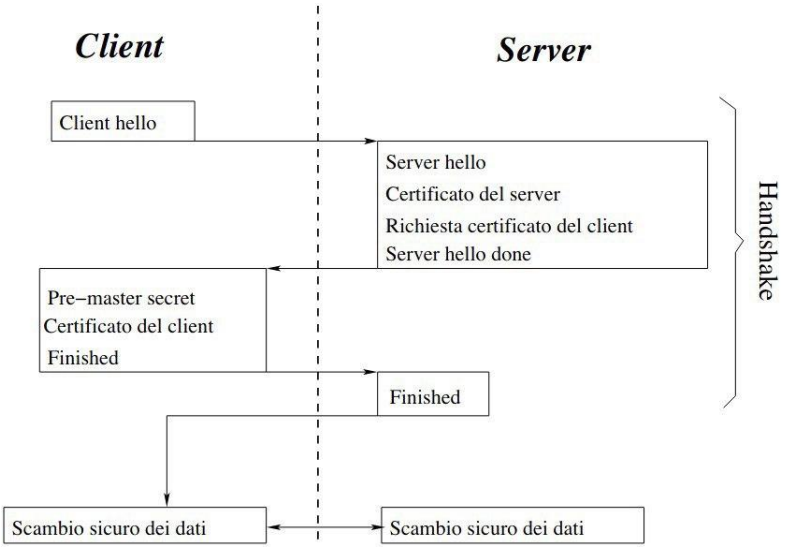
1. L'utente U manda a S un messaggio di **client hello** con cui richiede la creazione di una connessione SSL, specifica le "prestazioni" di sicurezza che desidera siano garantite durante la connessione, e invia una sequenza di byte casuali. Esempio: *cipher suite*

   `SSL_RSA_WITH_AES_CBC_SHA1`

   `RSA`, scambio chiavi di sessione
   `AES`, cifratura simmetrica
   `CBC`, impiego di un cifrario a composizione di blocchi
   `SHA1`, funzione hash one way per il MAC

2. Il sistema S riceve il messaggio di client hello e seleziona da questo uno cipher suite che anch'esso è in grado di supportare. Dopodiché invia all'utente un messaggio, detto **server hello** che specifica la sua scelta e contiene una nuova sequenza di byte casuali. Se U non riceve il messaggio di server hello, interrompe la comunicazione
3. Il sistema S si autentica con U inviandogli il proprio certificato digitale e gli eventuali altri certificati della catena di certificazione della sua CA fino alla CA radice. Se i servizi offerti da S devono essere protetti negli accessi, S può richiedere a U di autenticarsi inviando il suo certificato digitale. Avviene raramente perché la maggior parte degli utenti non ha un certificato personale. Di solito il sistema si accerta dell'identità dell'utente in un secondo tempo
4. Il sistema S invia il messaggio **hello done** con cui sancisce la fine degli accordi sulla cipher suite e sui parametri crittografici a essa associati
5. L'utente U per accertare l'autenticità del certificato ricevuto da S, controlla che la data corrente sia inclusa nel periodo di validità del certificato, che la CA che ha firmato il certificato sia tra quelle *fidate* e che la firma da essa apposta sia autentica
6. L'utente U costruisce un **pre-master secret** costituito da una nuova sequenza di byte casuali, lo cifra con il cifrario a chiave pubblica su cui si è accordato con S e spedisce il relativo crittogramma a S. 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. A tutte queste sequenze, l'utente U applica delle funzioni hash one-way secondo una combinazione opportuna. Il risultato costituisce il *master secret*
7. Il sistema S decifra il crittogramma contenente il pre-master secret ricevuto da U e calcola il master secret mediante le stesse operazioni eseguite dall'utente nel passo 6, in quanto dispone delle stesse informazioni
8. *Opzionale*: se all'utente U viene richiesto un certificato (passo 3) ed egli non lo possiede, 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: master secret e tutti i messaggi scambiati fino a quel momento. S controlla il certificato di U con operazioni simmetriche a quelle eseguite da U nel passo 5, e verifica l'autenticità e la correttezza della SSL history. In presenza di anomalie la comunicazioni con U viene interrotta
9. Il **messaggio finished** è il primo messaggio protetto mediante il master secret e la cipher suite su cui i due partner si sono accordati. Il messaggio viene prima costruito da U e spedito a S, poi costruito da S e spedito a U: nei due invii la struttura del messaggio è la stessa ma cambiano le informazioni in esso contenute. La sua costruzione avviene in due passi. All'inizio si concatenano il master secret, tutti i messaggi di hashshake scambiati fino a quel momento e l'identità del mittente (UoS). La stringa così ottenuta viene trasformata applicando le funzioni SHA-1 e MD5, dando origine a una coppia di valori che costituisce il messaggio finished. Questo messaggio è diverso nelle due comunicazioni poiché S aggiunge ai messaggi di handshake anche il messaggio finished ricevuto da U. Il destinatario della coppia (UoS) non può invertire la computazione precedente in quanto generata da funzioni one-way, ma ricostruisce l'ingresso delle due funzioni SHA1 e MD5, le ricalcola e controlla che la coppia generata coincida con quella ricevuta

Il master secret viene utilizzato da U e da S per costruire una propria tripla contenente la chiave segreta da adottare nel cifrario simmetrico, la chiave da adottare in una funzione hash one-way per la costruzione del MAC, e la sequenza di inizializzazione per cifrare in modo aperiodico messaggi molto lunghi. Le triple dell'U e di S sono diverse tra loro ma note a entrambi i partner: ciascuno usa la propria, il che aumenta la sicurezza delle comunicazioni. Il canale sicuro approntato dal protocollo SSL handshake viene realizzato dal protocollo SSL record. I dati sono frammentati in blocchi di lunghezza opportuna; ciascun blocco viene numerato, compresso, autenticato attraverso l'aggiunta di un MAC, cifrato mediante il cifrario simmetrico su cui l'utente e il sistema si sono accordati, e finalmente trasmesso dall'SSL record utilizzando il protocollo di trasporto sottostante. Il destinatario delle comunicazioni esegue un procedimento inverso sui blocchi ricevuti: decifra e verifica la loro integrità attraverso il MAC, decomprime e riassembla in blocchi in chiaro, infine li consegna all'applicazione sottostante.



9.8.1 Sicurezza

Criteri che hanno orientato la progettazione del controllo:

- **Client hello e server hello**
  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 impedisce a un crittoanalista di riutilizzare i messaggi di handshake catturati sul canale per sostituirsi a S in una successiva comunicazione con U

- **MAC dei blocchi di dati**
  SSL record numera in modo incrementale ogni blocco di dati e autentica il blocco 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 il contenuto del blocco, il numero del blocco nella sequenza, la chiave del MAC, e alcune stringhe note e fissati 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 due partner è difficile almeno quanto quello volto alla sua decrittazione

- **Il sistema è autenticato**
  Il canale definito da SSL handshake è immune da attacchi attivi man-in-the-middle poiché il sistema S viene autenticato con un certificato digitale. 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 decifrare quel crittogramma e costruisce il master secret, su cui si fonda la costruzione di tutte le chiavi segrete adottate nelle comunicazioni successive. Quindi solo il sistema S, ossia quello a cui si riferisce il certificato, potrà entrare nella comunicazione con l'utente U

- **L'utente può essere autenticato**
  Il certificato di U (se richiesto) e la sua firma apposta sui messaggi scambiati nel protocollo (SSL history) consentono a S di verificare che U sia effettivamente quello che dichiara di essere e che i messaggi siano stati effettivamente spediti da esso. L'opzionalità dell'autenticazione dell'utente ha reso l'SSL molto diffuso nelle transazioni commerciali via Internet: per gli utenti la necessità di certificazione può costituire un ostacolo pratico ed economico. L'utente può essere autenticato con altri metodi (login e password, numero carta di credito)

- **Generazione delle sequenze casuali**
  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. In particolare la sequenza corrispondente al pre-master secret viene generata da U e comunicata per via cifrata a S. La non predicibilità di questa sequenza è cruciale per la sicurezza del canale SSL: una sua cattiva generazione renderebbe il protocollo molto debole

- **Messaggio finished**
  Contiene tutte le informazioni scambiate nel corso dell'handshake. Lo scopo è quello di consentire ai due partner un ulteriore controllo sulle comunicazioni precedenti per garantire che queste siano avvenute correttamente, U e S dispongono dello stesso master secret e che la comunicazione non sia stata oggetto di un attacco attivo

Per concludere il protocollo SSL è sicuro almeno quanto il più debole cipher suite supportato. Poiché, dopo i tentativi di alcuni governi di limitare l'impiego della crittografia su Web per motivi di "sicurezza nazionale", dal Gennaio 2000 le norme internazionali non pongono alcuna limitazione sui cifrari impiegabili se non in alcuni paesi che si trovano in condizioni politiche particolari, è consigliabile disabilitare i propri sistemi dall'impiego di cifrari ormai notoriamente insicuri o di chiavi troppo corte.

Chapter 10

Quantum key exchange

10.1 Principi meccanica quantistica

Il progetto scientifico, per il quale è difficile prevedere quando e se sarà disponibile una realizzazione alla portata di tutti, è quello della costruzione di **computer quantistico**, termine che può riferirsi a macchine differenti che sfruttino i principi di quella branca della fisica. I principi della meccanica quantistica sono:

- **Sovrapposizione**, proprietà di un sistema quantistico di trovarsi in diversi stati contemporaneamente
- **Decoerenza**, la misurazione di un sistema quantistico disturba il sistema: il sistema disturbato perde la sovrapposizione degli stati e collassa in uno stato singolo
- **No-Cloning**, impossibilità di duplicare un sistema conservando nella copia lo stato quantistico dell'originale. È impossibile copiare uno stato quantistico non noto
- **Entanglement**, possibilità che due o più elementi si trovino in stati quantistici correlati tra loro in modo che, pur se portati a grande distanza, mantengono la correlazione

10.2 Protocollo BB84

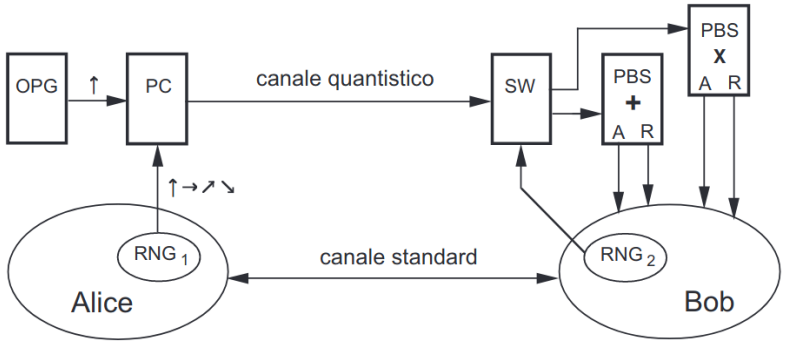
Nel 1984 Bennett e Brassard hanno definito il primo protocollo, noto come **BB84**, per lo scambio di chiavi tra due utenti mediante l'invio di fotoni polarizzati. Il **fotone** è una particella indivisibile priva di massa e si propaga alla velocità della luce. Ogni fotone possiede una **polarizzazione** definito come il piano di oscillazione del suo campo elettrico. La polarizzazione può essere misurata secondo due basi:

- **Base ortogonale** $+$, polarizzazione verticale $\mathcal{V}(\uparrow)$ e polirizzazione orizzontale $\mathcal{H}(\rightarrow)$
- **Base diagonale** $\times$, polarizzazione $+45°(\nearrow)$ e polarizzazione $-45°(\searrow)$

Questi quattro stati saranno associati ai bit 0 e 1 per rappresentare l'informazione codificata in una sequenza di fotoni, secondo le corrispondenze:

- **bit** $\mathbf{0}$, associato a un fotone polarizzato $\mathcal{V}(\uparrow)$ o $+45°(\nearrow)$
- **bit** $\mathbf{1}$, associato a un fotone polarizzato $\mathcal{H}(\rightarrow)$ o $-45°(\searrow)$

Se non si conosce in che base è stato polarizzato il fotone, non si possono distinguere tra le quattro polarizzazioni, ma solo tra due possibili polarizzazioni nella stessa base. Quindi non è possibile distinguere fra i 4 casi, l'unica misura possibile è tra due stati ortogonali nella stessa base.



Per realizzare il protocollo sono necessari tre componenti:

- **OPG**, consente di generare un fotone alla volta con una polarizzazione prestabilita
- **PC**, consente di imporre al fotone una certa polarizzazione
- **PBS**, determinano la polarizzazione del fotone e risponde correttamente solamente se la base di riferimento è la stessa del fotone (quindi scelta da Alice)

Alice sceglierà casualmente tramite l'**RNG1**, se mandare il bit 0 o 1, dopodiché, utilizzando il **PC**, sceglie una base casuale in cui preparare il fotone, quindi codificherà di conseguenza il fotone. Il fotone viaggia su fibra ottica. Alice non comunica a Bob la base, quindi quest'ultimo deve effettuare una scelta casuale per provare a indovinare in quale base è stato misura il fotone. Bob effettua questa scelta casuale tramite il deviatore **SW** che permette di scegliere, comandato da **RNG2**, se misurare con la base ortogonale o diagonale con **PBS**

Il PBS ha un suo asse di polarizzazione $\mathcal{S}$. Un fotone in ingresso polarizzato in direzione $\mathcal{F}$ viene inviato all'uscita A con probabilità $cos^2(\theta)$, o all'uscita R con probabilità $sin^2(\theta)$, dove θ è l'angolo compreso tra $\mathcal{S}$ e $\mathcal{F}$. Casi:

- $\boldsymbol{\theta = 0°}$ $\boldsymbol{(F = S)}$, il fotone esce da A con probabilità 1 e con polarizzazione $S(= F)$
- $\boldsymbol{\theta = 90°}$ $\boldsymbol{(F \perp S)}$, il fotone esce da R con probabilità 1 e con polarizzazione $S^{\perp}(= F)$
- $\boldsymbol{\theta = \pm 45°}$, il fotone esce con pari probabilità $1/2$ da A o da R, e la polarizzazione cambia (S o $S^{\perp}$)

Si riporta ora il funzionamento del protocollo:

Canale quantistico. Alice vuole trasmettere a Bob una sequenza $S_A[1, n]$, quindi per n volte sceglie una base a caso, codifica $S_A[i]$ e invia il fotone a Bob. Bob sceglie una base a caso, interpreta il fotone ricevuto e costruisce $S_B[i]$

Ora si passa al canale standard dove la comunicazione può essere anche in chiaro, l'importante che sia autenticata

Canale standard. Bob comunica ad Alice la sequenza di basi scelte, ed Alice comunica a Bob le basi che hanno in comune. Alice e Bob:

1. estraggono S'_A e S'_B corrispondenti alle basi comuni, quindi estraggono due sottosequenze di S'_A e S'_B in posizioni concordate: S''_A e S''_B.
2. si scambiano S''_A e S''_B, se la percentuale di bit diversi è maggiore della percentuale di errori dovuti all'apparato sperimentale (**QBER**) è segno dell'intervento del crittoanalista, quindi scartano la chiave e si fermano. Se invece la percentuale di errori è inferiore a questa soglia, calcolano S'_A/S''_A e S'_B/S''_B, le decodificano con un codice a correzione di errori e ottengono una sequenza comune: S_C
3. Calcolano l'immagine hash $K = h(S_C)$ e la usano come chiave

Se Eve intercetta i fotoni in viaggio da Alice verso Bob, si può comportare come Bob, ovvero misura il fotone con una base e lo invia Bob. Quindi Eve costruisce $S_E[i]$ non necessariamente per ogni i

Esempio 1. Simulazione protocollo senza Eve

S_A	$\mathbf{1}$	$\mathbf{0}$	$\mathbf{1}$	1	1	0	$\mathbf{0}$	...
Basi di Alice	$+$	$\times$	$+$	$\times$	$\times$	$+$	$\times$	...
Fotoni di Alice	$\rightarrow$	$\nearrow$	$\rightarrow$	$\searrow$	$\searrow$	$\uparrow$	$\nearrow$	...
Basi di Bob	$+$	$\times$	$+$	$+$	$+$	$\times$	$\times$	...
Fotoni di Bob	$\rightarrow$	$\nearrow$	$\rightarrow$	$\uparrow^*$	$\rightarrow^*$	$\searrow^*$	$\nearrow$	...
S_B	$\mathbf{1}$	$\mathbf{0}$	$\mathbf{1}$	0	1	1	$\mathbf{0}$	...

Alcuni fotoni di Bob hanno l'asterisco perché le basi sono discordi, per cui il fotone e il conseguente bit del risultato è casuale. Nella fase di controllo, Alice e Bob, conserveranno i fotoni misurati nella stessa base (in grassetto).

Esempio 2. Simulazione protocollo con la presenza di Eve

S_A	$\mathbf{1}$	$\mathbf{0}$	$\mathbf{1}$	1	1	0	$\mathbf{0}$	...
Basi di Alice	$+$	$\times$	$+$	$\times$	$\times$	$+$	$\times$	...
Fotoni di Alice	$\rightarrow$	$\nearrow$	$\rightarrow$	$\searrow$	$\searrow$	$\uparrow$	$\nearrow$	...
Basi di Eve	$+$	$+$	$+$	$\times$	$+$	$\times$	$+$	...
Fotoni di Eve	$\rightarrow$	$\rightarrow^*$	$\rightarrow$	$\searrow$	$\uparrow^*$	$\searrow^*$	$\rightarrow$	...
S_E	1	1	1	1	0	1	1	...
Basi di Bob	$+$	$\times$	$+$	$+$	$+$	$\times$	$\times$	...
Fotoni di Bob	$\rightarrow$	$\searrow^*$	$\rightarrow$	$\rightarrow^*$	$\uparrow$	$\searrow$	$\uparrow^*$	...
S_B	$\mathbf{1}$	$\boxed{\mathbf{1}}$	$\mathbf{1}$	1	0	1	$\mathbf{0}$	...

Adesso a seguito del controllo, nonostante le prime due basi di Alice coincidano con le due basi di Bob, i bit di risultato sono diversi. Quindi Alice e Bob si accorgono della presenza di Eve, e scartano la chiave