Crittografia

Alessio Delgadillo

Anno accademico 2020-2021

Indice

Capitolo 1

Introduzione

Con il termine **crittografia**, "*scrittura nascosta*", si intende lo studio di tecniche matematiche sofisticate per

mascherare i messaggi	crittografia
tentare di svelarli	crittoanalisi

La crittografia esiste perché esistono due mondi in contrapposizione: le persone che vogliono scambiarsi privatamente delle informazioni e gli "impiccioni" che desiderano ascoltare o intromettersi nelle conversazioni altrui per curiosità, investigazione, scopi malvagi... Di conseguenza le persone che vogliono rendere incomprensibili le proprie conversazioni ai non autorizzati sviluppano metodi di cifratura, mentre chi vuole riportare alla luce le informazioni contenute in quelle conversazioni sviluppa metodi di crittoanalisi.

Terminologia

crittografia	metodi di cifratura
crittoanalisi	metodi di interpretazione
crittologia	studio della comunicazione su canali non sicuri e relativi problemi

Lo scenario

Un agente Alice vuole comunicare con un agente Bob e deve utilizzare un canale di trasmissione **insicuro**, 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 in ascolto sul canale, ma facilmente decifrabile da Bob.

`MSG`: insieme dei messaggi (testi in chiaro)
`CRITTO`: insieme dei crittogrammi (testi cifrati)

Cifratura del messaggio: operazione con cui si trasforma un messaggio in un chiaro `m` in un crittogramma `c` applicando una funzione

$$C : \texttt{MSG} \to \texttt{CRITTO}$$

Decifratura del crittogramma: operazione che permette di ricavare il messaggio in chiaro `m` a partire dal crittogramma `c` applicando la funzione

$$D : \texttt{CRITTO} \to \texttt{MSG}$$

Le funzioni C e D sono una l'inversa dell'altra

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

Inoltre, la funzione C è iniettiva: *a messaggi diversi devono corrispondere crittogrammi diversi.*

1.1 Livello di segretezza

I metodi crittografici (cifrari) possono essere classificati in base al loro livello di segretezza: **cifrari per uso ristretto** e **cifrari per uso generale**.

1.1.1 Cifrari per uso ristretto

Le funzioni di *cifratura* C e di *decifrazione* D sono tenute **segrete** in ogni loro aspetto. Solitamente sono impiegati per comunicazione diplomatiche o militari e non sono adatti per una crittografia "di massa".

1.1.2 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 del metodo deve essere limitata a un'informazione aggiuntiva, la **chiave**, nota solo alla coppia di utenti che stanno comunicando. Le regole devono essere **pubbliche** e solo la chiave deve essere **segreta**:

il nemico conosce il sistema.

Chiave segreta

Una chiave segreta k deve essere

- diversa per ogni coppia di utenti,
- inserita come parametro nelle funzioni di cifratura e decifrazione.

$$c = C(m, k) \quad m = D(c, k)$$

Se non si conosce k, la conoscenza di C e D e del crittogramma non deve permettere di estrarre informazioni sul messaggio in chiaro.

- Tenere segreta la chiave è più semplice che tenere segreto l'intero processo di cifratura e decifrazione.
- Tutti possono impiegare le funzioni pubbliche C e D scegliendo chiavi diverse.
- Se un crittoanalista entra in possesso di una chiave occorre solo generarne una nuova, le funzioni C e D rimangono inalterate.
- Molti cifrari in uso (3DES, RC5, IDEA, AES) sono a chiave segreta.

Se la segretezza dipende unicamente dalla chiave, allora il numero delle chiavi deve essere praticamente immune ad ogni tentativo di provarle tutte e deve essere scelta in modo casuale.

Attacco esauriente

Il crittoanalista potrebbe sferrare un **attacco esauriente** verificando la significatività delle sequenze $D(c, k),\ \forall$ chiave k.

Se $|\text{Key}| = 10^{20}$ e un calcolatore impiegasse 10^{-6} secondi per calcolare $D(c, k)$ e verificarne la significatività, occorrerebbe più di un miliore di anni per scoprire il messaggio provando tutte le chiavi possibili.

Osservazioni

La segretezza può essere violata con altre tecniche di crittoanalisi.

Esistono cifrari più sicuri di altri, pur basandosi su uno spazio di chiavi molto più piccolo:

1. un cifrario complicato non è necessariamente un cifrario sicuro;
2. mai sottovalutare la bravura del crittoanalista.

1.2 Crittoanalista

- Comportamento passivo: Eve si limita ad ascoltare la comunicazione.
- Comportamento attivo: Eve agisce sul canale disturbando la comunicazione o modificando il contenuto del messaggio.

Gli attacchi ai sistemi crittografici hanno l'obiettivo generale di **forzare** il sistema. Metodo e livello di pericolosità dipendono dalle informazioni in possesso del crittoanalista:

- **Cipher Text Attack** (solo testo cifrato): il crittoanalista rileva sul canale una serie di crittogrammi $c_1, \ldots, c_r$.
- **Known Plain-Text Attack** (testo in chiaro noto): il crittoanalista conosce una serie di coppie $(m_1, c_1), \ldots, (m_r, c_r)$.
- **Choose Plain-Text Attack** (testo in chiaro scelto): il crittoanalista conosce una serie di coppie $(m_1, c_1), \ldots, (m_r, c_r)$ relative a messaggi in chiaro da lui scelti.

Attacchi "Man in-the-middle"

Il crittoanalista si installa sul canale di comunicazione, interrompe le comunicazioni dirette tra Alice e Bob, le sostituisce con messaggi propri e convince ciascun utente che tali messaggi provengono legittimamente dall'altro.

Eve si finge Alice agli occhi di Bob e Bob agli occhi di Alice.

Attacchi

L'attacco al sistema può essere portato a termine con pieno successo, si scopre la funzione D, o con successo limitato, si scopre solo qualche informazione su un particolare messaggio: l'informazione parziale può essere sufficiente per comprendere il significato del messaggio.

1.3 Situazione attuale

Cifrari perfetti (inattaccabili): esistono, ma richiedono operazioni estremamente complesse e sono utilizzati solo in condizioni estreme. Messaggio in chiaro e crittogramma risultano del tutto scorrelati tra loro: nessuna informazione sul testo può filtrare dal crittogramma e la conoscenze di Eve non cambia dopo aver osservato un crittogramma sul canale.

One-Time Pad: assolutamente sicuro, ma richiede una nuova chiave segreta per ogni messaggio che deve essere lunga come il messaggio da scambiare e perfettamente casuale.

Come si genera e come si scambia la chiave?

Estremamente attraente per chi richiede una sicurezza assoluta ed è disposto a pagarne i costi.

I cifrari diffusi in pratica non sono perfetti, ma sono **dichiarati sicuri** perché sono 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 venga la pena di sostenerlo, l'operazione richiede di impiegare giganteschi calcolatori per tempi incredibilmente lunghi: impossibilità pratica di forzare il cifrario.

Non è noto se la risoluzione di questi problemi matematici richieda **necessariamente** tempi enormi, oppure se i tempi enormi di risoluzione siano dovuti alla nostra **incapacità di individuare metodi più efficienti di risoluzione**.

1.3.1 Cifrari di oggi

Advanced **E**ncryption **S**tandard (**AES**):

- standard per le comunicazione riservate ma "non classificate";
- pubblicamente noto e realizzabile in hardware su computer di ogni tipo;
- chiavi brevi (qualche decina di caratteri, 128 o 256 bit).

Si tratta di un **cifrario simmetrico**, la stessa chiave è usata per cifrare e decifrare, e a blocchi, il messaggio è diviso in blocchi lunghi come la chiave: la chiave è utilizzata per trasformare un blocco del messaggio in un blocco del crittogramma.

Novità rispetto al passato: le chiavi segrete non sono stabilite direttamente dai partner (Alice e Bob), ma dai mezzi elettronici che utilizzano per comunicare; inoltre su Internet si costruisce una nuova chiave per ogni sessione.

Ma come si può scambiare una chiave segreta con facilità e sicurezza?

La chiave serve per comunicare in sicurezza, ma Alice e Bob devono stabilirla comunicando in "sicurezza" senza poter ancora usare il cifrario; un'intercettazione nell'operazione di scambio della chiave pregiudica il sistema.

Nel 1976 viene proposto (da Merkle, Hellman e Diffie) 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 D. e H. propongono alla comunità scientifica anche la definizione di **crittografia a chiave pubblica** (ma senza averne un'implementazione pratica): rivoluziona il modo di concepire le comunicazioni segrete; nata ufficialmente nel 1976 ma preceduta dal lavoro, coperto da segreto, degli agenti britannici (Ellis, Cock e Williamson).

Cifrari simmetrici

Nei cifrari simmetrici, la *chiave di decifratura è uguale a quella di decifrazione* (o l'una può essere facilmente calcolata dall'altra) ed è nota solo ai due partner che la scelgono di comune accordo e la mantengono segreta.

Cifrari a chiave pubblica (asimmetrici)

Obiettivo: permettere a tutti di inviare messaggi cifrati, ma abilitare solo il ricevente (Bob) a decifrarli. Le operazioni di cifratura e decifrazione sono pubbliche e utilizzano due chiavi **diverse**:

- $\mathrm{k_{pub}}$ per **cifrare**: è pubblica, nota a tutti;
- $\mathrm{k_{priv}}$ per **decifrare**: è privata, nota solo a Bob.

La **cifratura** di un messaggio m da inviare a Bob è eseguita da qualunque mittente come

$$c = C(m, k_{pub})$$

chiave k_{pub} e funzione di cifratura sono note a tutti.

La **decifrazione** è eseguita da Bob come

$$m = D(c, k_{priv})$$

la funzione di decifrazione è nota a tutti, ma k_{priv} non è disponibile agli altri.

La funzione di cifratura deve essere una funzione **one-way trap-door**: calcolare $c = C(m, k_{pub})$ è computazionalmente *facile*, decifrare c è computazionalmente *difficile* a meno che non si conosca un meccanismo segreto, rappresentato da k_{priv} (**trap-door**).

Nel 1977 Rivest, Shamir e Adleman propongono un sistema a chiave pubblica (RSA) basato su una funzione "facile" da calcolare e "difficile" da invertire.

Tutti gli utenti possono inviare in modo sicuro messaggi a uno stesso destinatario cifrandoli con la funzione C e la chiave k_{pub} che sono pubbliche; solo il destinatario può decifrare i messaggi. Un crittoanalista non può ricavare informazioni sui messaggi pur conoscendo C, D e k_{pub}.

Vantaggi

- Se gli utenti di un sistema sono n, il numero complessivo di chiavi (pubbliche e private) è $2n$ anziché $\frac{n(n-1)}{2}$.
- Non è richiesto alcuno scambio segreto di chiavi.

Svantaggi

- Questi sistemi sono **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 $m_1, \ldots, m_h$;
  - **cifrarli** con la funzione pubblica C e la chiave pubblica k_{pub} del destinatario, ottenendo i crittogrammi $c_1, \ldots, c_h$;
  - quindi può confrontare qualsiasi messaggio cifrato c' che viaggia verso il destinatario con i crittogrammi in suo possesso.

Cifrari ibridi

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.

La trasmissione dei messaggi lunghi avviene ad alta velocità, mentre è lento lo scambio delle chiavi segrete: le chiavi sono composte al massimo da qualche decina di byte e l'attacco chosen plain-text è risolto se l'informazione cifrata con la chiave pubblica (chiave segreta dell'AES) è scelta in modo da **risultare imprevedibile al crittoanalista**.

1.3.2 Applicazioni su rete

Oltre alla segretezza delle comunicazioni, i sistemi crittografici attuali devono garantire:

1. l'**identificazione** dell'utente: il sistema deve accertare l'identità di chi richiede di accedere ai suoi servizi.

2. l'**autenticazione** di un messaggio: Bob deve accertare che il messaggio ricevuto sia stato effettivamente spedito da Alice e deve poter stabilire che il messaggio non è stato modificato o sostituito durante la trasmissione.

3. la **firma digitale**: una volta apposta la "firma" Alice non può ricusare la paternità del messaggio spedito a Bob; Bob può dimostrare a terzi che il messaggio ricevuto è di Alice.

Altre applicazioni

- Trasmissione protetta di dati sulla rete (protocollo SSL).
- Moneta elettronica (BitCoin).
- Dimostrazioni a conoscenza zero.
- Applicazioni crittografiche dei fenomeni di meccanica quantistica.

Capitolo 2

Rappresentazione matematica e Teoria della Calcolabilità

2.1 Rappresentazione matematica di oggetti

Alfabeto: insieme finito di *caratteri* o *simboli*.

Un oggetto è rappresentato da una sequenza ordinata di caratteri dell'alfabeto:

- a oggetti diversi corrispondono sequenze diverse;
- il numero di oggetti che si possono rappresentare non ha limiti.

Alfabeti e sequenze

Alfabeto Γ, $|\Gamma| = s$

N oggetti da rappresentare.

- $d(s, N)$: lunghezza della sequenza più lunga 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 a $d_{\min}(s, N)$.

Rappresentazione unaria

$s = 1, \Gamma = \{0\}$: le sequenze di rappresentazioni possono essere solo ripetizioni dello 0

$$\Rightarrow d_{\min} = (1, N) = N$$

rappresentazione estremamente sfavorevole $0, 00, \ldots, 00000$, etc.

Rappresentazione binaria

$s = 2, \Gamma = \{0, 1\}$: $\forall k \geq 1,\ 2^k$ sequenze di lunghezza k. Il numero totale di sequenze lunghe da 1 a k è dato da

$$\sum_{i=0}^{k} 2^i = 2^{k+1} - 2$$

N oggetti da rappresentare

$$\Rightarrow 2^{k+1} - 2 \geq N$$

$$\Rightarrow k \geq \log_2(N + 2) - 1$$

$d_{\min}(2, N)$: minimo intero k che soddisfa tale relazione

$$\Rightarrow d_{\min}(2, N) = \lceil \log_2(N + 2) - 1 \rceil$$

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

$\lceil \log_2 N \rceil$ caratteri binari sono sufficienti per costruire N sequenze differenti:

$$N = 7, \quad \lceil \log_2 7 \rceil = 3$$
$$0, 1, 00, 01, 10, 11, 000$$

Si possono costruire N sequenze differenti tutte di $\lceil \log_2 N \rceil$ caratteri:

$$N = 7$$
$$000, 001, 010, 011, 100, 101, 110$$

Rappresentazioni s-arie

Si possono costruire N sequenze differenti con $\lceil \log_s N \rceil$ caratteri; si possono costruire N sequenze differenti tutte di $\lceil \log_s N \rceil$ caratteri.

Per esempio: $\Gamma = \{0, 1, 2, ..., 9\}$ con $\lceil \log_{10} 1000 \rceil = 3$ caratteri, si ottengono 1000 sequenze (dalla 000 alla 999).

Rappresentazioni

Usare sequenze della stessa lunghezza è vantaggioso: per concatenare sequenze di lunghezza diversa è necessario inserire una marca di separazione tra l'una o l'altra; questi accorgimenti non sono necessari se la lunghezza delle sequenze è unica e nota.

Rappresentazioni efficienti: rappresentazioni che usano un numero massimo di caratteri di ordine *logaritmico* nella cardinalità N dell'insieme da rappresentare; l'alfabeto deve contenere almeno 2 caratteri.

Rappresentazioni di interi

La notazione posizionale per rappresentare i numeri interi è una rappresentazione efficiente, indipendentemente dalla base $s \geq 2$ scelta. Un intero N è rappresentato con un numero d di cifre tale che

$$\lceil \log_s N \rceil \leq d \leq \lceil \log_s N \rceil + 1$$

Riduzione logaritmica tra il valore N di un numero e la lunghezza d della sua rappresentazione.

2.2 Teoria della Calcolabilità

Si occupa delle questioni fondamentali circa la **potenza** e le **limitazioni** dei sistemi di calcolo. L'origine risale alla prima metà del ventesimo secolo, quando i logici matematici iniziarono ad esplorare i concetti di computazione, algoritmo e problema risolvibile per via algoritmica e dimostrarono l'esistenza di **problemi che non ammettono un algoritmo di risoluzione**: *problemi non decidibili.*

Problemi computazionali

Problemi formulati matematicamente di cui cerchiamo una soluzione algoritmica. Classificazione:

- problemi non decidibili
- problemi decidibili
  - problemi trattabili (costo polinomiale)
  - problemi intrattabili (costo esponenziale)

Calcolabilità	nozioni di *algoritmo* e di *problema non decidibile*
Complessità	nozioni di *algoritmo efficiente* e di *problema intrattabile*

La calcolabilità ha lo scopo di classificare i problemi in **risolvibili** e **non risolvibili**, mentre la complessità in "*facili*" e "*difficili*".

2.2.1 Esistenza di problemi indecidibili

Insiemi numerabili

Due insiemi A e B hanno lo stesso numero di elementi se e solo se si può stabilire una *corrispondenza biunivoca* tra i loro elementi.

Un insieme è **numerabile** (possiede una infinità numerabile di elementi) se e solo se i suoi elementi possono essere messi in *corrispondenza biunivoca con i numeri naturali*. Un insieme numerabile è un insieme i cui elementi possono essere enumerati, ossia descritti da una sequenza del tipo $a_1, a_2, \ldots, a_n, \ldots$

Enumerazione delle sequenze

Si vogliono elencare in un ordine ragionevole le sequenze di lunghezza finita costruite su un alfabeto finito; le sequenze non sono in numero finito, quindi non si potrà completare l'elenco. Scopo:

- raggiungere qualsiasi sequenza σ arbitrariamente scelta in un *numero finito di passi*;
- σ deve dunque trovarsi a distanza finita dall'inizio dell'elenco.

Ordinamento canonico: 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". Una sequenza s arbitraria si troverà tra quelle di $|s|$ caratteri, in posizione alfabetica tra queste.

- Ad una sequenza arbitraria corrisponde il numero che ne indica la posizione nell'elenco.
- A un numero naturale n corrisponde la sequenza che occupa la posizione n nell'elenco.

Osservazione: la numerazione delle sequenze è possibile perché esse sono di **lunghezza finita** anche se illimitata, cioè per un qualunque intero d scelto a priori esistono sequenze di lunghezza maggiore di d. Per sequenze di lunghezza infinita la numerazione non è possibile.

Gli insiemi non numerabili sono tutti gli insiemi non equivalenti a N:

- insieme dei numeri reali
- insieme dei numeri reali compresi nell'intervallo aperto (0,1)
- insieme dei numeri reali compresi nell'intervallo chiuso [0,1]
- insieme di tutte le linee nel piano
- insieme delle funzioni in una o più variabili

L'insieme dei problemi computazionali NON è numerabile.

2.2.2 Problemi e funzioni

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

$$f : N^k \to N^j$$

L'insieme delle funzioni $f : N^k \to N^j$ NON è numerabile

Diagonalizzazione

$F = \{\text{funzioni } f \mid f : N \to \{0,1\}\}$ ogni $f \in F$ può essere rappresentata da una sequenza infinita

$$\begin{array}{ll} x & 0\ 1\ 2\ 3\ 4\ \ldots n\ \ldots \\ f(x) & 0\ 1\ 0\ 1\ 0\ \ldots 0\ \ldots \end{array}$$

o, se possibile, da una regola finita di costruzione

$$f(x) = \begin{cases} 0 & x \textit{ pari} \\ 1 & x \textit{ dispari} \end{cases}$$

Teorema 2.2.1. *L'insieme F non è numerabile.*

Dimostrazione. Per assurdo, F sia numerabile. Possiamo enumerare ogni funzione: assegnare ad ogni $f \in F$ un numero progressivo nella numerazione e costruire una tabella (infinita) di tutte le funzioni.

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	...
...	...	...

Consideriamo la funzione $g \in F$

$$g(x) = \begin{cases} 0 & f_x(x) = 1 \\ 1 & f_x(x) = 0 \end{cases}$$

g non corrisponde ad alcuna delle f_i della tabella: **differisce da tutte nei valori posti sulla diagonale principale**.

Per assurdo $\exists\ j$ t.c. $g(x) = f_j(x)$ allora $g(j) = f_j(j)$, ma per definizione

$$g(j) = \begin{cases} 0 & f_j(j) = 1 \\ 1 & f_j(j) = 0 \end{cases}$$

contraddizione!

Per qualunque numerazione scelta, esiste sempre almeno una funzione esclusa: F non è numerabile. Si possono considerare linee arbitrarie che attraversano la tabella toccando tutte le righe e tutte le colonne esattamente una volta, e definire funzioni che assumono in ogni punto un valore opposto a quello incontrato sulla linea. Esistono infinite funzioni di F escluse da qualsiasi numerazione. □

Conclusione

$F = \{f : N \to \{0, 1\}\}$ non è numerabile; a maggior ragione, non sono numerabili gli insiemi delle funzioni

$$f : N \to N$$
$$f : N \to R$$
$$f : R \to R$$
$$f : N^k \to N^j$$

L'insieme dei problemi computazionali non è numerabile.

2.2.3 Il problema della rappresentazione

L'informatica rappresenta tutte le sue entità (quindi anche gli algoritmi) in forma digitale, come *sequenze finite di simboli di alfabeti finiti* (e.g. {0,1})

Definizione 2.2.1. Un *algoritmo* è una sequenza finita di operazioni, completamente e univocamente determinate.

La formulazione di un algoritmo dipende dal modello di calcolo utilizzato

- programma per un modello matematico astratto, come una Macchina di Turing
- algoritmo per in pseudocodice per RAM
- programma in linguaggio C per un PC

Qualsiasi modello si scelga, gli algoritmi devono esservi descritti, ossia rappresentati da sequenze finite di caratteri di un alfabeto finito; di conseguenza *gli algoritmi sono possibilmente infiniti*, ma numerabili: possono essere "elencati" (messi in corrispondenza biunivoca con l'insieme dei numeri naturali).

Drastica perdita di potenza: gli algoritmi sono numerabili e sono meno dei problemi computazionali, che hanno la potenza del continuo

$$|\{\text{Problemi}\}| >> |\{\text{Algoritmi}\}|$$

Esistono problemi privi di un corrispondente algoritmo di calcolo.

2.2.4 Il problema dell'arresto

Esistono dunque problemi non calcolabili. I problemi che si presentano spontaneamente sono tutti calcolabili, non è stato facile individuare un problema che non lo fosse.

Il **problema dell'arresto** (Turing, 1930) è un problema posto in forma decisionale

$$\text{Arresto} : \{Istanza\} \rightarrow \{0, 1\}$$

Per i problemi decisionali, la calcolabilità è in genere chiamata decidibilità.

Presi ad arbitrio un algoritmo A e i suoi dati di input D, decidere in ***tempo finito*** *se la computazione di A su D termina o no.*

Si tratta di un algoritmo che indaga sulle proprietà di un altro algoritmo, trattato come dato di input. È legittimo: possiamo usare lo stesso alfabeto per

codificare algoritmi e i loro dati di ingresso (sequenze di simboli dell'alfabeto); una stessa sequenza di simboli può essere quindi interpretata sia come un programma, sia come un dato di ingresso di un altro programma.

Il problema consiste nel chiedersi se un generico programma termina la sua esecuzione oppure va in "ciclo", ovvero continua a ripetere la stessa sequenza di istruzioni all'infinito (supponendo di non avere limiti di tempo e memoria).

Turing ha dimostrato che riuscire a dimostrare se un programma arbitrario si arresta e termina la sua esecuzione non è solo un'impresa ardua, ma in generale è impossibile!

Teorema 2.2.2. *Il problema dell'arresto è indecidibile.*

Dimostrazione. Se il problema dell'arresto fosse decidibile, allora esisterebbe un algoritmo `ARRESTO` che, presi A e D come dati di input, determina in tempo finito le risposte

$$\texttt{ARRESTO}(A, D) = \begin{cases} 1 & \text{se } A(D) \text{ termina} \\ 0 & \text{se } A(D) \text{ non termina} \end{cases}$$

Osservazione: l'algoritmo `ARRESTO` non può consistere in un algoritmo che simuli la computazione $A(D)$; se A non si arresta su D, `ARRESTO` non sarebbe in grado di rispondere NO (0) in tempo finito.

In particolare possiamo scegliere $D = A$, cioè considerare la computazione $A(A)$:

$$\texttt{ARRESTO}(A, A) = 1 \Leftrightarrow A(A) \text{ termina}$$

Se esistesse `ARRESTO`, esisterebbe anche il seguente algoritmo

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

L'ispezione dell'algoritmo `PARADOSSO` mostra che

$$\texttt{PARADOSSO}(A) \text{ termina} \Leftrightarrow x = \texttt{ARRESTO}(A, A) = 0 \Leftrightarrow A(A) \text{ non termina}$$

Cosa succede calcolando `PARADOSSO(PARADOSSO)`?

$$\texttt{PARADOSSO}(\texttt{PARADOSSO}) \text{ termina}$$

$$\Leftrightarrow x = \texttt{ARRESTO}(\texttt{PARADOSSO}, \texttt{PARADOSSO}) = 0$$

$$\Leftrightarrow \texttt{PARADOSSO}(\texttt{PARADOSSO}) \text{ non termina}$$

L'unico modo di risolvere la contraddizione è che l'algoritmo `PARADOSSO` non possa esistere, dunque non può esistere nemmeno l'algoritmo `ARRESTO`.

□

Osservazione: non può esistere un algoritmo che decida in tempo finito se un algoritmo arbitrario termina la sua computazione su dati arbitrari; ciò non significa che non si possa decidere in tempo finito la terminazione di algoritmi particolari. Il problema è indecidibile su una coppia $\langle A, D \rangle$ scelta arbitrariamente.

L'algoritmo `ARRESTO` costituirebbe uno strumento estremamente potente: permetterebbe infatti di dimostrare congetture ancora aperte sugli interi, per esempio la congettura di Goldbach.

"**Lezione di Turing**": *non esistono algoritmi che decidono il comportamento di altri algoritmi esaminandoli dall'esterno, cioè senza passare dalla loro simulazione*

2.2.5 Modelli di calcolo

La teoria della calcolabilità dipende dal modello di calcolo, oppure la decidibilità è una proprietà del problema?

I linguaggi di programmazione esistenti sono tutti equivalenti? Ce ne sono alcuni più potenti e/o più semplici di altri? Ci sono algoritmi descrivibili in un linguaggio, ma non in un altro? È possibile che problemi oggi irrisolvibili possano essere risolti in futuro con altri linguaggi o con altri calcolatori? La teorie della calcolabilità e della complessità dipendono dal modello di calcolo?

Tesi di Church-Turing

Tutti i (ragionevoli) modelli di calcolo ***risolvono esattamente la stessa classe di problemi***, dunque si equivalgono nella possibilità di risolvere problemi, pur operando con diversa efficienza.

La decidibilità è una proprietà del problema.

Incrementi qualitativi alla struttura di una macchina, o alle istruzioni di un linguaggio di programmazione, servono solo a abbassare il tempo di esecuzione e rendere più agevole la programmazione.

Capitolo 3

Teoria della Complessità

Decidibilità e Trattabilità

Ci sono problemi (problema dell'arresto) che non possono essere risolti da nessun calcolatore, indipendentemente dal tempo a disposizione: problemi indecidibili. Ci sono problemi decidibili che possono richiedere tempi di risoluzione esponenziali nella dimensione dell'istanza (torri di Hanoi, generazione delle sequenze binarie e delle permutazioni): **problemi intrattabili**. Ci sono problemi che possono essere risolti con algoritmi di costo polinomiale (ordinamento; ricerca di chiavi in array, liste, alberi; problemi su grafi: OT di DAG, connettività, ricerca di cicli, ricerca di un ciclo euleriano): **problemi trattabili** ("facili"). Ci sono infine problemi il cui stato non è noto (clique, cammino hamiltoniano); abbiamo a disposizione solo algoritmi di costo esponenziale, ma nessuno ha dimostrato che non possano esistere algoritmi di costo polinomiale: **problemi presumibilmente intrattabili**.

Studiamo la dimensione dei dati trattabili in funzione dell'incremento della velocità dei calcolatori.

Calcolatori: C_1, C_2 (k volte più veloce di C_1).
Tempo di calcolo a disposizione: t

- n_1 = dati trattabili nel tempo t su C_1
- n_2 = dati trattabili nel tempo t su C_2

Osservazione: usare C_2 per un tempo t, equivale a usare C_1 per un tempo $k \cdot t$.

Algoritmo **polinomiale** che risolve il problema in cn^s secondi (c, s costanti)

$$C_1 : \ cn_1^s = t \rightarrow n_1 = \left(\frac{t}{c}\right)^{\frac{1}{s}}$$

$$C_2 : \ cn_2^s = kt \rightarrow n_2 = \left(\frac{kt}{c}\right)^{\frac{1}{s}} = k^{\frac{1}{s}} \left(\frac{t}{c}\right)^{\frac{1}{s}}$$

$$n_2 = k^{\frac{1}{s}} n_1$$

Miglioramento di un fattore moltiplicativo $k^{\frac{1}{s}}$.

Algoritmo **esponenziale** che risolve il problema in $c2^n$ secondi (c costante)

$$C_1 : \ c2^{n_1} = t \rightarrow 2^{n_1} = \frac{t}{c}$$

$$C_2 : \ c2^{n_2} = kt \rightarrow 2^{n_2} = \frac{kt}{c} = k2^{n_1}$$

$$n_2 = n_1 + \log_2 k$$

Miglioramento di un fattore moltiplicativo $\log_2 k$.

3.1 Problemi

Problema Π

- I: insieme delle **istanze** in ingresso
- S: insieme delle **soluzioni**

Problemi decisionali: richiedono una risposta binaria ($S = \{0, 1\}$)

Istanze positive (accettabili):	$x \in I$, t.c. $\Pi(x) = 1$
Istanze negative:	$x \in I$, t.c. $\Pi(x) = 0$

Problemi di ricerca: data un'istanza x, richiedono di restituire una soluzione s.
Problemi di ottimizzazione: data un'istanza x, si vuole trovare la migliore soluzione s tra tutte le soluzioni possibili.

Problemi decisionali

La teoria della complessità computazionale è definita principalmente in termini di *problemi di decisione*: essendo la risposta binaria, non ci si deve preoccupare del tempo richiesto per restituire la soluzione e tutto il tempo è speso esclusivamente per il calcolo. La difficoltà di un problema è già presente nella sua versione decisionale.

Molti problemi di interesse pratico sono però problemi di ottimizzazione: è possibile esprimere un problema di ottimizzazione in forma decisionale,

chiedendo l'esistenza di una soluzione che soddisfi una certa proprietà. Il problema di ottimizzazione è **almeno tanto difficile quanto** il corrispondente problema decisionale.

Caratterizzare la complessità del problema decisionale permette quindi di dare almeno una **limitazione inferiore** alla complessità del problema di ottimizzazione.

3.1.1 Classi di complessità

Dato un problema decisionale Π ed un algoritmo A, diciamo 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 $s(n)$.

Classi Time e Space

Data una qualunque funzione $f(n)$

- ***Time(f(n))***: insiemi dei *problemi decisionali* che possono essere risolti in tempo $O(f(n))$.
- ***Space(f(n))***: insiemi dei *problemi decisionali* che possono essere risolti in spazio $O(f(n))$.

Classe P

Algoritmo polinomiale (tempo): esistono due costanti $c, n_0 > 0$ t.c. il numero di passi elementari è al più n^c per ogni input di dimensione n e $\forall n > n_0$. La classe P è la classe dei problemi **risolvibili in tempo polinomiale** nella dimensione n dell'istanza di ingresso.

Classe PSpace

Algoritmo polinomiale (spazio): esistono due costanti $c, n_0 > 0$ t.c. il numero di celle di memoria utilizzate è al più n^c per ogni input di dimensione n e $\forall n > n_0$. La classe PSpace è la classe dei problemi **risolvibili in tempo polinomiale** nella dimensione n dell'istanza di ingresso.

Classe Exp-Time

La classe Exp-Time è la classe dei problemi **risolvibili in tempo esponenziale** nella dimensione n dell'istanza di ingresso.

Relazioni tra classi

$$\text{P} \subseteq \text{PSpace}$$

infatti un **algoritmo polinomiale** può avere accesso al più ad un numero polinomiale di locazioni di memoria diverse (in ordine di grandezza).

$$\text{PSpace} \subseteq \text{ExpTime}$$

Non è noto (ad oggi) se le inclusioni siano proprie, l'unico risultato di separazione dimostrato finora riguarda P e ExpTime: esiste un problema che può essere risolto in tempo esponenziale, ma per cui tempo polinomiale non è sufficiente (per esempio le "Torri di Hanoi").

Algoritmo per Clique

Si considerano tutti i sottoinsiemi di vertici, in ordine di cardinalità decrescente, e si verifica se formano una clique di dimensione almeno k. Se n è il numero di vertici, al caso peggiore l'algoritmo esamina 2^n sottoinsiemi diversi.

$$\text{Clique} \in \text{ExpTime}$$

Algoritmo polinomiale non noto.

Algoritmo per Cammino Hamiltoniano

Si considerano tutte le permutazioni di vertici, e si verifica se i vertici in quell'ordine sono a due a due adiacenti. Se n è il numero di vertici, al caso peggiore l'algoritmo esamina $n!$ permutazioni diverse.

$$\text{CamminoHamiltoniano} \in \text{ExpTime}$$

Algoritmo polinomiale non noto.

SAT

Insieme V di variabili booleane

- Letterale: variabile o sua negazione.

- Clausola: disgiunzione (OR) di letterali.

Un'espressione booleana su V si dice in forma normale congiuntiva (FNC) se è espressa come *congiunzione di clausole* (AND di OR di letterali), per esempio:

$$V = \{x, y, z, w\} \qquad FNC : (x \vee \neg y \vee z) \wedge (\neg x \vee w) \wedge y$$

Data una espressione in forma normale congiuntiva verificare se esiste una assegnazione di valori di verità alle variabili che rende l'espressione vera. L'algoritmo per SAT considera tutti i 2^n assegnamenti di valore alle n variabili, e per ciascuno verifica se la formula è vera.

$$\text{SAT} \in \text{ExpTime}$$

Algoritmo polinomiale non noto.

La ricerca esaustiva è necessaria? Non lo sappiamo.

3.1.2 Problemi decisionali e certificati

In un *problema decisionale* siamo interessati a verificare se una istanza del problema soddisfa una certa proprietà.

- Esiste una clique di k vertici?
- Esiste un cammino hamiltoniano?
- Esiste un assegnamento di valori che rende vera la formula?

Per alcuni problemi, per le **istanze accettabili** (positive) x è possibile fornire un **certificato** y che possa convincerci del fatto che l'istanza soddisfa la proprietà e dunque è un'istanza accettabile.

Certificato

- Certificato per Clique: sottoinsieme di k vertici, che forma la clique.
- Certificato per Cammino Hamiltoniano: permutazione degli n vertici che definisce un cammino semplice.
- Certificato per SAT: un'assegnazione di verità alle variabili che renda vera l'espressione.

Un certificato è un **attestato breve di esistenza** di una soluzione con determinate proprietà. Si definisce solo per le istanza *accettabili*, infatti, in generale non è facile costruire *attestati di non esistenza*.

Verifica

Idea: **utilizzare il costo della verifica** di un certificato (una soluzione) per un'istanza accettabile (positiva) per **caratterizzare la complessità** del problema stesso.

Un problema Π è verificabile in tempo polinomiale se:

1. ogni istanza accettabile x di Π di lunghezza n ammette un **certificato y di lunghezza polinomiale** in n;
2. esiste un **algoritmo di verifica polinomiale** in n e applicabile a ogni coppia $\langle x, y \rangle$ che permette di attestare che x è accettabile.

3.1.3 Classe NP

NP è la classe dei problemi decisionali **verificabili in tempo polinomiale**. Cosa vuol dire NP? P sta per polinomiale, ma N non vuol dire NON... La classe NP è la classe dei problemi risolvibili in tempo ***polinomiale non deterministico***.

Osservazione: un certificato contiene un'informazione molto prossima alla soluzione, quindi qual è l'interesse di questa definizione? Dubbio legittimo:

- La teoria della verifica è utile per far luce sulle gerarchie di complessità dei problemi, non aggiunge nulla alla possibilità di risolverli efficientemente.
- Chi ha una soluzione può verificare in tempo polinomiale che l'istanza è accettabile.
- Chi non ha una soluzione (certificato), può individuarla in tempo esponenziale considerando tutti i casi possibili con una ricerca esaustiva.

Le classi P e NP

$$P \subset NP$$

Ogni problema in P ammette un certificato verificabile in tempo polinomiale: eseguo l'algoritmo che risolve il problema per costruire il certificato.

$$P = NP \text{ oppure } P \neq NP?$$

Non sappiamo se per risolvere un problema della classe NP dobbiamo per forza passare attraverso una ricerca esaustiva oppure no. Si congettura che $P \neq NP$. È possibile individuare i problemi più difficili all'interno della classe NP, ovvero quelli candidati ad appartenere a NP se $P \neq NP$.

3.1.4 Problemi NP-Completi

Sono i problemi più difficili all'interno della classe NP: 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 $P = NP$. Quindi tutti i problemi NP-Completi sono risolvibili in tempo polinomiale oppure nessuno lo è.

Riduzioni polinomiali

Π_1 e Π_2 sono problemi decisionali, I_1 e I_2 insiemi delle istanze di input di Π_1 e Π_2. Π_1 *si riduce in tempo polinomiale* a Π_2

$$\Pi_1 \leq_p \Pi_2$$

se esiste una funzione $f : I_1 \to I_2$ calcolabile in tempo polinomiale t.c. $\forall x \in \Pi_1$, x è un'istanza accettabile di Π_1 se e solo se $f(x)$ è un'istanza accettabile di Π_2.

Se esistesse un algoritmo per risolvere Π_2 potremmo utilizzarlo per risolvere Π_1

$$\Pi_1 \leq_p \Pi_2 \text{ e } \Pi_2 \in P \Rightarrow \Pi_1 \in P$$

Un problema Π si dice **NP-arduo** se

$$\forall\ \Pi' \in NP, \Pi' \leq_p \Pi$$

Un problema decisionale Π si dice **NP-Completo** se

$$\Pi \in NP \text{ e } \forall\ \Pi' \in NP, \Pi' \leq_p \Pi$$

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-arduo o NP-completo: bisogna dimostrare che TUTTI i problemi in NP si riducono polinomialmente a Π. In realtà la prima dimostrazione di NP-completezza aggira il problema.

Teorema 3.1.1 (Cook, 1971)**.** *SAT è NP completo.*

Cook ha mostrato che dati un qualunque problema $\Pi \in NP$ 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.

Un problema decisionale Π è NP-completo se

$$\Pi \in NP \qquad \text{SAT} \leq_p \Pi$$

(o un qualsiasi altro problema NP-completo); infatti,

$$\forall\ \Pi' \in NP,\ \Pi' \leq_p \text{SAT e SAT} \leq_p \Pi$$

Quindi $\Pi' \leq_p \Pi$.

$$\text{SAT} \leq_p \text{Clique}$$

data un'espressione booleana F in forma normale congiuntiva con k clausole è possibile costruire in *tempo polinomiale* un grafo G che contiene una **clique di k vertici se esolo se F è soddisfacibile**.

$$\text{SAT} \leq_p \text{Clique} \Rightarrow \text{Clique è NP-Completo}$$

$$\text{SAT è NP-Completo} \Rightarrow \text{Clique} \leq_p \text{SAT}$$

SAT e Clique sono **NP-equivalenti**. Tutti i problemi NP completi sono tra loro NP-equivalenti; sono tutti facili, o tutti difficili.

Problemi di ottimizzazione NP-arduo

Se la soluzione ottima è troppo difficile da ottenere, una soluzione quasi ottima ottenibile facilmente forse è buona abbastanza. A volte, avere una soluzione **esatta** non è strettamente necessario: ci si accontenta di una soluzione che non si discosti troppo da quella ottima e che si possa calcolare in tempo polinomiale.

3.1.5 Le classi Co-P e Co-NP

Profonda differenza tra certificare l'esistenza o la non-esistenza di una soluzione, per esempio nel problema del ciclo Hamiltoniano:

- una permutazione di vertici (per certificare esistenza)
- per la non-esistenza è difficile dare un certificato polinomiale che indichi direttamente questa proprietà

Π: problema decisionale.
coΠ: problema *complementare* (accetta tutte e sole le istanze rifiutate da P.

La classe co-P è la classe dei **problemi decisionali** Π per cui

$$\text{co}\Pi \in P$$

P = co-P (risolvo il problema, e complemento il risultato).

Si congettura che NP e co-NP siano diverse; se questa congettura è vera, allora $P \neq NP$.

Capitolo 4

Teoria della casualità

In crittografia c'è un gran bisogno di sequenze casuali: la generazione della chiave deve essere imprevedibile, inoltre spesso si deve ricorrere ad algoritmi randomizzati.

4.1 Significato algoritmico della casualità

Si prendano due sequenze di lunghezza n

$$\begin{array}{ll} h: & 1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\ldots1 \\ h': & 1\,0\,1\,1\,0\,1\,1\,0\,1\,0\,1\,1\ldots0 \end{array}$$

Per descrivere la prima sequenza basterebbe dire che è composta da n volte "1"; per descrivere la seconda si dovrebbe dettarla.

La **casualità** è una *proprietà di quelle sequenze che* ***non*** *possono essere descritte in modo compatto.* Formalizziamo la suddetta definizione:

$$P(h) = \left(\frac{1}{20}\right)^{20} \qquad P(h') = \left(\frac{1}{20}\right)^{20}$$

L'algoritmo A_h che genera la sequenza h sarebbe $\langle$`genera n 1`$\rangle$ e, immaginandolo di averlo codificato in binario, la sua lunghezza in bit sarebbe

$$|A_h| = \Theta(\log n)$$

Intuizione: una sequenza binaria è casuale se non ammette un algoritmo di generazione la cui rappresentazione binaria sia più corta della sequenza stessa.

Difatti, A'_h è del tipo $\langle$`print "cifra1"`, `print "cifra2"`, $\ldots\rangle$, perciò, il numero di bit necessari a codificarlo sarà sicuramente uno per ogni cifra della sequenza insieme alle costanti per le keyword del linguaggio. Perciò, $|A'_h| \geq |n| = |h'|$.

Sistemi di calcolo

Sono un'infinità numerabile $S_1, S_2, \ldots, S_i, \ldots$ Si prenda come riferimento il sistema di calcolo S_i e si supponga che p sia un programma che genera una sequenza generica h nel sistema S_i.

Definizione 4.1.1. La complessità di Kolmogorov di h nel sistema S_i è

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

In altre parole, la complessità di h nel sistema S_i è la lunghezza del programma più breve che genera h nel sistema S_i.

Osservazione: se la sequenza h non obbedisce ad alcuna legge semplice, allora il più corto programma che la può generare deve contenerla al suo interno e la genera trasferendola in output.

$$K_{S_i}(h) = |h| + c_i$$

dove c_i è una costante che dipende da S_i e rappresenta la parte di programma che trasferisce h in output. Tuttavia si tratta di una definizione vincolata a sistemi di calcolo specifici, perciò è necessario considerare un *sistema universale* per poterla svincolare.

Il **sistema di calcolo universale** S_u è in grado di simulare qualsiasi altro sistema di calcolo; in particolare, preso p come un programma che genera h nel sistema S_i, allora $S_u(\langle i, p\rangle)$ è il sistema S_u che simula S_i con al suo interno il programma p:

$$S_u(\langle i, p\rangle) = S_i(p) = h$$

È possibile considerare $q = \langle i, p\rangle$ come il programma che genera h in S_u. Quanto è grande q?

$$|q| = |\langle i, p\rangle| = |i| + |p| = \log_2 i + |p|$$

q dipende da i, ma non da h. Si può affermare che

$$\forall h \; K_{S_u}(h) \leq K_{S_i}(h) + c$$

il simbolo "=" si tiene di conto quando non esistono sistemi migliori da simulare (che permettono di trovare h) oltre che S_i, mentre il simbolo $<$ si tiene di conto quando esiste un altro sistema $S_j \neq S_i$ che contiene un programma che restituisce h più corto di quello presente in S_i.

Definizione 4.1.2. La complessità di Kolmogorov di una sequenza h è

$$K(h) = K_{S_u}(h)$$

Definizione 4.1.3. Una sequenza h è casuale se

$$K(h) \geq |h| - \lceil \log_2 |h| \rceil$$

Osservazione: la casualità è un proprietà della sequenza (non dipende dalla sorgente che ha generato la sorgente).

4.1.1 Due risultati della teoria di Kolmogorov

Conteggio delle sequenze (positivo)

$$\forall n,\ \exists \text{ sequenze casuali (secondo Kolmogorov) di lunghezza } n.$$

Dimostrazione. Sia $S = 2^n$ il numero di sequenze binarie di lunghezza n e T il numero di sequenze di lunghezza n non casuali. Obiettivo: $T < S$. Per far questo si consideri N, il numero di sequenze binarie di lunghezza $< n - \lceil log_2(n) \rceil$ che equivale a:

$$\sum_{i=0}^{n-\lceil \log n \rceil - 1} 2^i = 2^{n-\lceil \log n \rceil} - 1$$

Tra queste N sequenze ci sono anche i programmi che generano le T sequenze non casuali di lunghezza n. Da questo calcolo ricaviamo:

$$T \leq N < S \Rightarrow T < S$$

Quindi, $\forall n$, esistono certamente sequenze casuali di lunghezza n e sono enormemente più numerose di quelle non casuali.

$$\frac{T}{S} \leq \frac{N}{S} = \frac{2^{n-\lceil (\log n) \rceil} - 1}{2^n} = \frac{1}{2^{\lceil \log n \rceil}} - \frac{1}{2^n} < \frac{1}{2^{\lceil \log n \rceil}} \Rightarrow \lim_{n \to \infty} \frac{T}{S} = 0$$

□

Stabilire se una sequenza arbitaria è casuale (negativo)

Secondo Kolmogorov stabilire se una sequenza arbitraria è casuale è un problema indecidibile.

Dimostrazione. Supponiamo che esista un algoritmo `Random` t.c.

$$\texttt{Random}(h) = \begin{cases} 1 & \text{se } h \text{ casuale} \\ 0 & \text{altrimenti} \end{cases}$$

Allora possiamo costruire l'algoritmo `Paradosso` che enumera tutte le possibili sequenze binarie in ordine di lunghezza crescente.

```
Paradosso:
  for binary h = 1 → ∞ do{
    if(|h| − ⌈log |h|⌉ > |p| ∧ Random(h) == 1)
      return h;
}
```

p è una sequenza binaria che rappresenta `Paradosso` e `Random`, ovvero rappresenta il programma complessivo: $|p| = |\texttt{Paradosso}| + |\texttt{Random}|$ e $|p|$ non dipende da h perché essa appare come nome di variabile (il suo valore è registrato in una cella di memoria al di fuori del programma).

Quindi, `Paradosso` restituisce come risultato la prima sequenza casuale h di lunghezza t.c. $|h| - \lceil \log_2 |h| \rceil > |p|$; esistendo sequenze casuali di qualsiasi lunghezza, certamente ne esisterà una che soddisfi entrambe le condizioni. Tuttavia la prima condizione afferma che p è breve e genera h, quindi h non può essere casuale, mentre la seconda condizione afferma il contrario.

Il programma `Random` non può esistere.

□

Capitolo 5

Generatore di sequenze pseudocasuali

Una **sorgente binaria casuale** genera una sequenza di bit con le seguenti caratteristiche:

1. $P(0) = P(1) = \frac{1}{2}$ (si può indebolire richiedendo $P(0), P(1) > 0$ e immutabili durante il processo di generazione);
2. la generazione di un bit è indipendente da quella degli altri bit: non si può prevedere il valore di un bit osservando quelli già generati.

Si supponga di avere $P(0) > P(1)$, è sempre possibile bilanciare la sequenza: si generi una sequenza come

$$0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0$$

Si dividono i bit a coppie e si scartano le coppie con bit uguali:

$$1\ 0\ 0\ 1$$

e associamo alle coppie discordi un valore: $01 \to 0,\ 10 \to 1$

Nella pratica è impossibile garantire la perfetta casualità di una sorgente e, soprattutto, l'indipendenza; non si può garantire che la generazione di un bit avvenga in modo assolutamente indipendente dalla generazione di altri bit: ogni esperimento modifica l'ambiente, quindi si va a influenzare l'esperimento successivo. Perciò, si accettano compromessi.

Ci sono varie tecniche per generare sequenze casuali brevi. È possibile sfruttare alcuni **fenomeni casuali** presenti in natura (decadimento di particelle, ...), ma il problema di questo approccio è che nessuno deve avere

accesso ai dispositivi utilizzati per sfruttare queste sorgenti; inoltre più il fenomeno è lungo, più tende a ripetersi periodicamente e quindi è accettabile solo per sequenze casuali molto brevi. Può essere sfruttata anche la casualità presente in alcuni **processi software**, come la posizione della testina sull'hard disk, lo stato dell'orologio interno del calcolatore, ...

5.1 Generatori di numeri pseudocasuali

Si genera la casualità mediante un algoritmo, cercandola all'interno di processi matematici: si parla di **generatori pseudo-casuali** perché sono programmi brevi e, quindi, per Kolmogorov non sono casuali. I generatori di numeri pseudo-casuali partono da una sequenza piccola chiamata *seme* (che può essere generata con processi come i due citati sopra) e generano una sequenza molto più lunga.

Un **generatore** prende in input un *seme* e restituisce in output un flusso di bit arbitrariamente lungo e periodico: al suo interno contiene una sottosequenza che si ripete (*periodo*) ed è quella che viene usata come sequenza casuale; un generatore è tanto migliore tanto è più lungo il periodo. Si tratta di un "*amplificatore di casualità*": prende la casualità all'interno del seme iniziale e la "estende", la "amplifica".

Limitazioni

Si supponga che S sia il numero di bit del seme e che si generi una sequenza di lunghezza n; tipicamente $n >> S$. Una volta che il seme di S bit è stato scelto, la sequenza lunga n che ne deriva è già determinata implicitamente dal seme; questo significa che due semi uguali forniscono due sequenze uguali. Inoltre, il numero di sequenze diverse che possono essere generate con un seme da S bit sono 2^S (meno di tutte quelle che si potrebbero costruire: 2^n), proprio perché questo è il numero di semi che si possono avere con S bit: ecco perché si parla di generatore *pseudocasuale*.

Esempio di generatore

Generatore lineare: $x_i = (ax_{i-1} + b)\ mod\ m$, dove a, b e m sono interi positivi. Il seme è un valore intero iniziale x_0 (che potremmo pensare di aver estratto casualmente). Si osservi che, poiché lavora col modulo, questo generatore può generare i numeri da 0 a $m - 1$:

$$x_0,\ x_1 = (ax_0 + b)\ mod\ m,\ x_2 = (ax_1 + b)\ mod\ m,\ \ldots$$

È chiaro che nel momento in cui $x_i = x_0$ la sequenza si ripeterà perché ricomincerà a generare gli stessi numeri. Quindi il periodo massimo è proprio m, ma non è garantito che lo sia sempre; per far ciò bisogna imporre certe condizioni sui parametri:

- $\text{MCD}(b, m) = 1$;
- $(a - 1)$ deve essere divisibile per ogni fattore primo di m;
- $(a - 1)$ deve essere un multiplo di 4 se anche m lo è.

Se si rispettano queste condizioni il periodo sarà m e quindi verranno generate permutazioni degli interi da 0 a $m - 1$. Esempio:

$$a = 7,\ b = 7,\ m = 9,\ x_0 = 3$$

$$x_i = (7x_{i-1} + 7)\ mod\ 9$$

$$x_0 = 3,\ x_1 = 1,\ x_2 = 5,\ x_3 = 7,\ x_4 = 2,\ \ldots,\ x_i = 3$$

Nota bene: per trasformarlo in generatore binario si fa $\frac{x_i}{m}$, si guarda la prima cifra decimale e se è pari allora si ottiene uno 0, altrimenti si ottiene un 1.

Valutare un generatore

Le sequenze generate da un generatore si valutano tramite dei **test statistici**, in particolare valutano se la sequenze presenta proprietà tipiche di una sequenza casuale.

- **Test di frequenza**: controlla che i vari simboli diversi abbiano la stessa frequenza.
- **Poker test**: controlla che le sottosequenze di lunghezza fissa occorrano lo stesso numero di volte nella sequenza totale.
- **Test di autocorrelazione**: verifica che non ci siano irregolarità nella sequenza, per esempio, guardando quali sono gli elementi che si ripetono dopo una certa lunghezza (ad esempio: ogni cinque posti c'è uno 0).
- **Run test**: verifica che le sottosequenze massimali di elementi tutti ripetuti abbiano una decrescenza esponenziale di apparizione nella sequenza in base a quanto sono lunghe (11 probabile, 111 poco probabile, 1111 quasi impossibile, 11111 impossibile, ...).

Per le applicazioni crittografiche si richiede anche il **test di prossimo bit** (implica anche i test precedenti): controlla che sia impossibile fare previsioni in tempo polinomiale sugli elementi della sequenza prima che siano generati. In particolare, un generatore binario supera il test di prossimo bit se non esiste un algoritmo polinomiale in grado di prevedere l'$(i+1)$-esimo bit della sequenza a partire dalla conoscenza degli i-esimi bit precedenti con probabilità maggiore di $\frac{1}{2}$. I generatori che superano il test di prossimo bit sono detti **crittograficamente sicuri**.

Generatore polinomiale

Non è crittograficamente sicuro.

$$x_i = (a_1 x_{i-1}^t + a_2 x_{i-1}^{t-1} + \ \cdots + a_t x_{i-1} + a_{t+1}) \ mod \ m$$

5.1.1 Generatori crittograficamente sicuri

Si fa ricorso alle **funzioni one-way**

- compututazionalmente facile da calcolare: $x \rightarrow y = f(x)$ richiede tempo polinomiale.
- compututazionalmente difficile da invertire: $y \rightarrow x = f^{-1}(y)$ richiede tempo esponenziale.

Idea: si supponga di avere un seme x_0 dal quale si ottiene un valore x; si calcola $f(x)$ considerando f come funzione one-way e si trova x_1. In seguito si trova x_2 con $f(x_1) = f(f(x)) = f^2(x)$. In generale, $x_i = f^i(x) = f(x_{i-1})$: si itera l'applicazione della funzione one-way un numero arbitrario di volte.

Ogni elemento della sequenza S si può calcolare facilmente dal precedente, ma non dai valori successivi perché f è one-way; di conseguenza, si calcola la sequenza per un certo numero di passi, senza svelarne il risultato, e si comunicano/utilizzano gli elementi in ordine inverso: ogni elemento non è prevedibile in tempo polinomiale, pur conoscendo quelli comunicati in precedenza.

Per costruire generatori binari crittograficamente sicuri si usano i predicati "*hard core*" delle funzioni one-way. $b(x)$ è un predicato hard core di una funzione one-way $f(x)$ se:

- $b(x)$ è facile da calcolare conoscendo x;
- $b(x)$ è difficile da prevedere con probabilità $> \frac{1}{2}$ conoscendo solo $f(x)$.

Esempio di funzione one-way

Funzione one-way: $f(x) = x^2 \ mod \ n$, n non primo. Supponiamo $n = 77$ e $x = 10$:

- $f(x) = 10^2 \ mod \ 77 = 23 \rightarrow$ molto facile.
- $x = ? \rightarrow$ molto difficile: tipicamente si fa un forza bruta (costoso) dove si provano tutti i numeri da 0 a $n - 1$ e quando troviamo quello che corrisponde a $f(x)$ (23 in questo caso) ci fermiamo e guardiamo a quale x corrisponde.

Nota bene: è importante ricordare che nell'algebra modulare, la funzione di elevamento a quadrato (x^2) è one-way. In questo caso il predicato hard core $b(x)$ è "x è dispari".

Generatore BBS (1986)

Si prende un numero $n = p \cdot q$, con p, q primi e molto grandi; inoltre

$$p \ mod \ 4 = 3 \text{ e } q \ mod \ 4 = 3 \qquad 2\left\lfloor \frac{p}{4} \right\rfloor + 1 \text{ e } 2\left\lfloor \frac{q}{4} \right\rfloor + 1 \text{ primi tra loro}$$

e y coprimo con n.

Si calcola il seme come $x_0 = y^2 \ mod \ n$ e una successione di $m \leq n$ interi

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

Successivamente si pone $b_i = 1 \Leftrightarrow x_{n-1}$ è dispari.

Il problema del generatore BBS è che per mantenersi sicuro deve usare numeri molto grandi ed effettuare potenze e moduli, perciò ha computazioni molto lente.

Generatori basati su cifrari simmetrici

Si prende un cifrario simmetrico e la chiave e si sostituisce il messaggio con un valore iniziale legato al generatore.

Esempio approvato dal FIPS: usa il DES.

- $r =$ numero di bit delle parole prodotte ($r = 64$ in DES)
- $s =$ seme casuale di r bit
- $m =$ numero di parole da produrre

- k = chiave segreta del cifrario

```
Generatore(s, m){ //flusso di output di m*r bit
    d = rappresentazione su r bit di data e ora
    y = C(d, k);
    z = s;
    for(i = 1; i <= m; i++){
        x_i = C(y XOR z, k);
        z = C(y XOR x_i, k);
        comunica x_i a chi di dovere;
    }
}
```

Capitolo 6

Algoritmi randomizzati

Sono algoritmi alimentati non solo dai dati del problema ma anche da una sequenza di valori casuali così da poter compiere passi randomici. Si dividono in due classi:

- **Las Vegas**: generano un risultato *sicuramente corretto* in un tempo *probabilmente breve* (quicksort).
- **Monte Carlo**: generano un risultato *probabilmente corretto* in un tempo *sicuramente breve* (test di primalità) e impongono che la probabilità di errore sia arbitrariamente piccola e matematicamente misurabile.

6.1 Test di primalità (Miller e Rabin, '80)

N è un numero intero dispari di n bit da testare. $N-1$ è pari e si può rappresentare come $2^w z$, dove z è dispari e w è l'esponente della potenza di 2 più grande che divide $N-1$: esempio

$$N = 17,\ N-1 = 16 = 2^4 \cdot 1 \quad \rightarrow \quad z = 1, w = 4$$

Si osservi che z e w si calcolano in tempo polinomiale nel numero di cifre perché è possibile dividere N per 2 al massimo $\lceil \log_2 N \rceil$ prima di trovare 1 (nel caso peggiore in cui $N =$ potenza di 2).

Sia N un numero primo e $2 \leq y \leq N-1$:

1. $\text{MCD}(N, y) = 1$;
2. $y^z \ mod\ N = 1$ oppure $\exists i,\ 0 \leq i \leq w-1$ t.c. $y^{2^i z} \ mod\ N = -1$

Si osservi che entrambe sono condizioni sufficienti: se un numero è primo allora soddisfa le due suddette condizioni, ma esistono numeri non primi che le soddisfano. Fortunatamente quei numeri non primi che soddisfano i predicati sono pochi e ciò conferisce al metodo una probabilità di errore molto bassa.

Lemma 1 (Miller, Rabin)

Se N è un numero composto, il numero di interi compresi fra 2 e $N-1$ che soddisfano entrambi i predicati è minore di $\frac{N}{4}$.

Dato un intero N e scelto un y a caso che appartiene a $[2, N-1]$:

- Se uno dei due predicati è falso, allora N è certamente composto.
- Se i predicati sono entrambi veri, allora N è composto con probabilità $< \frac{1}{4}$, dunque N è primo con probabilità $> \frac{3}{4}$.

Una probabilità di errore del 25% è troppo alta. Per ridurla si itera k volte e in questo modo la probabilità di errore diventa $< \left(\frac{1}{4}\right)^k$.

Algoritmo del test di primalità

```
Verifica(N, y){ //controlla la validità del certificato y
    if(P1 == F or P2 == F) return 1;
    //certificato valido: N è certamente composto
    else return 0;
    //certificato falso: N è probabilmente primo (errore < 1/4)
}

TestMR(N, k){
    for(i = 1; i <= k; i++){
        scegli a caso y in [2, N - 1]
        if(Verifica(N, y) == 1) return 0; //N è composto
        scegli nuovo y
    }
    return 1; //N è probabilmente primo (P < k/4)
}
```

Costo del test di primalità

`TestMR` ripete k volte `Verifica`, vediamo quanto costa `Verifica`. Esso va a valutare i due predicati e il primo è un MCD, non costa granché, mentre il

secondo è un po' più costoso, quindi vediamone il costo. È necessario calcolare

$$y^z \ mod \ N = 1 \text{ oppure } \exists i, \ 0 \leq i \leq w-1 \text{ t.c. } y^{2^i z} \ mod \ N = -1$$

L'esponente massimo per y si ottiene quando $i = w-1 \Rightarrow y^{\frac{N-1}{2}} \ mod \ N$.

Per avere un algoritmo polinomiale si vuole eseguire al massimo $\log N$ moltiplicazioni, quindi si usa l'algoritmo *delle quadrature successive* (*esponenziazione veloce*). Immaginiamo di dover calcolare $x = y^z \ mod \ s$ con x, y, s dello stesso ordine di grandezza.

1. Si scompone l'esponente z in una somma di potenze di 2:

$$z = \sum_{i=0}^{t} k_i \cdot 2^i \qquad k_i \in \{0, 1\}$$

   Si noti che $t = \lfloor \log_2 z \rfloor = \Theta(\log z)$.

2. Si calcolano tutte le potenze

$$y^{2^i} \ mod \ s \qquad 1 \leq i \leq t = \lfloor \log_2 z \rfloor$$

   ciascuna come il quadrato della precedente:

$$y^{2^i} \ mod \ s = \left(y^{2^{i-1}}\right)^2 \ mod \ s$$

Esempio:

$$x = 9^{45} \ mod \ 11 \qquad 45 = 32 + 8 + 4 + 1$$
$$y^2 \ mod \ s = 9^2 \ mod \ 11 = 4$$
$$y^4 \ mod \ s = 4^2 \ mod \ 11 = 5$$
$$y^8 \ mod \ s = 5^2 \ mod \ 11 = 3$$
$$y^{16} \ mod \ s = 3^2 \ mod \ 11 = 9$$
$$y^{32} \ mod \ s = 9^2 \ mod \ 11 = 4$$

A questo punto calcoliamo $x = y^z \ mod \ s$:

$$x = \prod_{\{i \ | \ k_i \neq 0\}} y^{2^i} \ mod \ s$$

Esempio:

$$y^z \ mod \ s = 9^{45} \ mod \ 11 = 9^{32+8+4+1} \ mod \ 11 =$$
$$= \left(\left(9^{32} \ mod \ 11\right) \cdot \left(9^8 \ mod \ 11\right) \cdot \left(9^4 \ mod \ 11\right) \cdot \left(9^1 \ mod \ 11\right)\right) \ mod \ 11 =$$
$$= (4 \cdot 3 \cdot 5 \cdot 9) \ mod \ 11 = 1$$

Costo:

- $\Theta(\log_2 z)$ quadrature.
- $O(\log_2 z)$ moltiplicazioni.

Ogni moltiplicazione ha un costo al più quadratico nel numero di cifre e quindi si ottiene un algoritmo polinomiale nella dimensione dei dati.

6.2 Generazione di numeri primi

Si genera un numero intero casuale dispari, seguito dal test di primalità e si ripete fino a quando si trova un numero *dichiarato* primo. Gli interi da testare in realtà sono pochi a causa della **densità dei numeri primi sull'asse degli interi**: il numero di interi primi e minori di N sono

$$\frac{N}{\log N} \text{ per } N \to \infty$$

Per N sufficientemente grande, in un suo intorno di ampiezza $\log N$ cade mediamente un numero primo e questo significa che testiamo al massimo una quantità logaritmica, ovvero *polinomiale* nel numero delle cifre di N, di numeri prima di trovarne uno primo.

```
Primo(n, k) //genera un numero primo almeno n bit
    S = sequenza binaria di n-2 bit generata pseudocasualmente
    N = concatenazione(1, S, 1)         //N contiene n bit
    while(TestMR(N, k) == 0) N = N + 2  //10 in binario
    return N
```

Si noti che $P_{err} < \left(\frac{1}{4}\right)^k$.

`TestMR` ha costo polinomiale in $n = \log N$, in particolare è $O(n^3)$, e il corpo del `while` viene ripetuto $O(n)$ volte, quindi è un algoritmo complessivamente polinomiale.

6.3 Classe RP (Random Polinomial)

Si tratta della classe dei problemi decisionali **verificabili in tempo polinomiale randomizzato**.

Preso un problema Π con l'istanza di input x, si definisce un certificato probabilistico y di x che ha una lunghezza al più polinomiale in $|x|$ ed è estratto perfettamente a caso da un insieme associato ad x. Questo certificato è utile se associato a un algoritmo di verifica polinomiale: `A(x, y)` attesta

con certezza, in *tempo polinomiale*, che x **non** possiede la proprietà ($\Pi(x) = 0$) oppure attesta che x possiede la proprietà esaminata dal problema con probabilità $> \frac{1}{2}$.

$$P \subset RP \subset NP$$

Si congettura che

$$P \subseteq RP \subseteq NP$$

Capitolo 7

Cifrari storici

Lo scopo dei cifrari storici era quello di consentire comunicazioni "sicure" tra poche persone, ma sono stati tutti forzati. La cifratura e la decifrazioni erano realizzate con carta e penna, e i messaggi da cifrare erano frasi di senso compiuto in linguaggio naturale.

I cifrari storici seguivano i **principi di Bacone** (XIII secolo):

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

7.1 Cifrario di Cesare

È il più antico cifrario di concezione moderna. Il crittogramma c è ottenuto dal messaggio in chiaro m sostituendo ogni lettera di m con quella tre posizioni più avanti nell'alfabeto.

Si noti che non vi è alcuna chiave segreta: la segretezza dipende dalla sola conoscenza del metodo, scoprire il metodo significava comprometterne irrimediabilmente l'impiego. Per questi motivi, il cifrario di Cesare era destinato all'**uso ristretto** di un gruppo di conoscenti.

Può essere trasformato aumentandone la sicurezza: invece di ruotare l'alfabeto di tre posizioni, è possibile ruotarlo di una quantità arbitraria k, $1 \leq k \leq 25$ (26 lascia inalterato il messaggio); in questo caso k è la **chiave segreta** del cifrario.

Formulazione matematica

$pos(X)$ è la posizione della lettera X nell'alfabeto e k è la chiave, $1 \leq k \leq 25$.

- Cifratura di X: lettera Y tale che $pos(Y) = (pos(X) + k)\ mod\ 26$
- Decifrazione di Y: lettera X tale che $pos(X) = (pos(Y) - k)\ mod\ 26$

Realizzazione fisica

Due dischi concentrici

- disco interno: lettere dell'alfabeto in chiaro;
- disco esterno: lettere cifrate.

Si pone una lettera del disco interno in corrispondenza con una lettera del disco esterno, in accordo con k. Ovviamente, tutte le altre lettere combaceranno con le corrispondenti lettere cifrate.

Crittoanalisi

Se si conosce la struttura del cifrario, si possono applicare in breve tutte le chiavi possibili (25) a un crittogramma per decifrarlo e scoprire contemporaneamente la chiave segreta: cifrario inutilizzabile a fini crittografici.

Osservazioni

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 modificarne il crittogramma finale.

Comporre più cifrari non aumenta la sicurezza del sistema: date due chiavi k_1, k_2 e una sequenza s

$$C(C(s, k_2), k_1) = C(s, k_1 + k_2)$$

$$D(D(s, k_2), k_1) = D(s, k_1 + k_2)$$

una sequenza di operazioni di cifratura e decifrazione può essere ridotta ad una sola operazione di cifratura o decifrazione.

7.2 Classificazione dei cifrari storici

Il cifrario di Cesare è detto **a sostituzione**: sostituisce ogni lettera del messaggio in chiaro con una o più lettere dell'alfabeto secondo una regola prefissata. Si possono distinguere due classificazioni

- Sostituzione **monoalfabetica**: alla stessa lettera del messaggio corrisponde sempre una stessa lettera nel crittogramma (e.g. cifrario di Cesare). Nei cifrari di questo tipo è possibile impiegare funzioni di cifratura più complesse della semplice addizione e sottrazione in modulo. Questo rende lo spazio delle chiavi più ampio ma la loro sicurezza rimane comunque molto modesta. Sono stati tutti forzati grazie alla crittoanalisi statistica che studia l'ordine, la frequenza e la posizione con cui si presentano parole e lettere.

- Sostituzione **polialfabetica**: alla stessa lettera del messaggio corrisponde una lettera scelta in un insieme di lettere possibili secondo una regola opportuna, ad esempio, a seconda della posizione della lettera da sostituire oppure del contesto in cui essa appare nel messaggio.

Esistono anche i cifrari **a trasposizione** che permutano le lettere del messaggio in chiaro secondo una regola prefissata.

7.2.1 Cifrari a sostituzione monoalfabetica

Cifrario affine

Cifratura: una lettera in chiaro X viene sostituita con la lettera cifrata Y che occupa nell'alfabeto la posizione

$$pos(Y) = (a \cdot pos(X) + b) \ mod \ 26$$

$k = \langle a, b \rangle$: chiave segreta del cifrario.

Decifrazione:

$$pos(X) = a^{-1} \cdot (pos(Y) - b) \ mod \ 26$$

a^{-1}: è l'inverso di a modulo 26:

$$a \cdot a^{-1} = 1 \ mod \ 26$$

L'inverso di un intero a modulo m **esiste ed è unico** se e solo se

$$\text{MCD}(a, m) = 1$$

Si tratta di un vincolo forte sulla definizione della chiave: se $\text{MCD}(a, m) \neq 1$, la *funzione di cifratura non è iniettiva e la decifrazione diventa impossibile.*

a e 26 devono essere co-primi

- i fattori primi di 26 sono 2 e 13

- a può assumere qualsiasi valore dispari tra 1 e 25, ad eccezione di 13 (12 valori possibili)

b può essere scelta liberamente tra 0 e 25, dunque può assumere 26 valori.

Le chiavi legittime sono tutte le possibili $\langle a, b \rangle$, in totale $12 \times 26 = 312$ chiavi (troppo poche); in realtà sono 311 (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 ad ogni tentativo di provarle tutte. La chiave segreta deve essere scelta in modo (possibilmente) casuale da un insieme molto grande.

In un cifrario a sostituzione monoalfabetica è possibile aumentare la cardinalità delle chiavi prendendo una permutazione arbitraria dell'alfabeto come chiave:

lettera in chiaro di posizione $i \Rightarrow$ lettera di posizione i nella permutazione.

In questo caso, lo spazio delle chiavi **aumenta enormemente**: abbiamo una chiave fatta di 26 lettere invece dei due numeri interi del cifrario affine; di conseguenza, lo spazio delle chiavi è esteso a $26! - 1$ ($\sim 4 \cdot 10^{26}$) chiavi e quindi è vasto e inesplorabile con metodi esaurienti. Tuttavia, il cifrario non è ancora sicuro: il sistema è attaccabile facilmente con un'**analisi statistica** sulla frequenza dei caratteri.

7.2.2 Cifrario a sostituzione polialfabetica

Archivio cifrato di Augusto

Augusto manteneva il proprio archivio cifrato: i documenti erano scritti in numeri, non in lettere. Fu svelato dall'imperatore Claudio circa 30 anni dopo la sua morte.

Augusto scriveva i suoi documenti in greco, poi metteva in corrispondenza la sequenza di lettere del documento con la sequenza di lettere del primo libro dell'Iliade; dopodiché, sostituiva ogni lettera del documento con il numero che indicava la distanza, nell'alfabeto greco, di tale lettera con quella in pari posizione dell'Iliade.

Esempio:

- lettera in posizione i-esima nel documento: α
- lettera in posizione i-esima nell'Iliade: ε

- carattere in posizione I-esima nel crittogramma: 4 (distanza fra α e ε)

Osservazioni:

- cifrario difficile da forzare se la chiave è lunghissima;
- utilizzato nella seconda guerra mondiale prendendo come chiave una pagina prefissata di un libro e cambiandola di giorno in giorno;
- lo svantaggio è che la chiave va messa per iscritto e inviata al destinatario (nel caso in cui ci sia).

Cifrario di Leon Battista Alberti

Disco di Leon Battista Alberti (XV secolo):

- **alfabeto esterno**: lettere (alcune) e numeri, per formulare il messaggio.
- **alfabeto interno**: più ricco, disposto in modo arbitrario (e diverso per ogni coppia di utenti), per costruire il crittogramma.

L'allineamento iniziale dei dischi avviene prima e in segreto. I successivi allineamenti vengono scoperti dal destinatario mano a mano che decifra (corrispondenza con un numero).

Metodo "indice mobile"

In questo metodo si usano sempre i dischi dell'Alberti ma si usa una tecnica particolare: quando si inserisce il numero n, che di norma rappresenterebbe un cambio di chiave, qui indica che dopo n caratteri la chiave verrà cambiata.

Osservazioni:

- si cambia chiave ogni volta che si incontra un carattere speciale;
- inserendo spesso i caratteri speciali (scartati nel messaggio ricostruito) il cifrario è difficile da attaccare;
- il continuo cambio di chiave rende inutili gli attacchi basati sulla frequenza dei caratteri.

La **Macchina Enigma** (1918) è un'estensione elettromeccanica del cifrario di Alberti.

Cifrario di Vigenère (1586)

La chiave è corta e ripetuta ciclicamente. Ogni lettera della chiave indica una *traslazione della corrispondente lettera del testo.*

Cifratura: si dispongono m e k su due righe adiacenti, allineando le lettere in verticale (se k è più corta di m, la chiave si ricopia più volte). Ogni lettera X del messaggio in chiaro risulta allineata ad una lettera Y della chiave. La X viene sostituita nel crittogramma con la lettera che si trova nella cella di T all'incrocio tra la riga che inizia con X e la colonna che inizia con Y.

Risulta essere un cifrario simmetrico, perciò necessita che le chiavi siano scambiate privatamente fra mittente e destinatario. 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**.

Per ogni intero positivo $i \leq h$ si costruisce un sottomessaggio $m[i]$ formato dalle lettere di m che occupano le posizioni $i, i+h, i+2h, \ldots$ In ciascuno di tali sottomessaggi tutte le lettere sono allineate con la stessa lettera della chiave. Il messaggio decomposto in h sottomessaggi, ciascuno dei quali è di fatto **cifrato con un metodo monoalfabetico**. I cifrari polialfabetici non sono dunque tanto più potenti dei monoalfabetici se le chiavi non sono molto lunghe.

One-Time Pad (1917)

Se estendiamo il metodo di Vigenère impiegando una chiave *lunga come il testo, casuale e non riutilizzabile*, il cifrario diviene **inattaccabile**: non può essere decifrato senza conoscere la chiave. È il caso di **One-Time Pad** che impiega un codice binario per messaggi e chiavi: fu usato nella *Hot Line* per le comunicazioni tra la Casa Bianca e il Cremlino a partire dal 1967.

Tuttavia, se la chiave è lunga quanto il messaggio, non può essere riutilizzata e necessita di essere scambiata fra i due estremi in maniera privata, tanto vale scambiarsi il messaggio. Difatti, il grosso problema di questi cifrari è proprio la lunghezza della chiave.

7.2.3 Cifrari a trasposizione

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.

Cifrario a permutazione semplice

La chiave è costituita da un intero h e da una permutazione π degli interi $\{1, 2, \ldots, h\}$. 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 (*padding*) che partecipano alla trasposizione, ma sono ignorate dal destinatario perché la decifrazione le riporta alle fine del messaggio.

Osservazione: il numero delle chiavi è

$$h! - 1$$

h non è fissato; 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.

Cifrario a permutazione di colonne

$k = \langle c, r, \pi \rangle$

- c e r denotano il numero di colonne e di righe di una tabella di lavoro T.
- π: permutazione degli interi $\{1, 2, \ldots, c\}$.

Il messaggio m viene decomposto in blocchi $m_1, m_2, \ldots$ di $c \times r$ caratteri ciascuno. Per cifrare il messaggio, i caratteri sono distribuiti tra le celle di T in modo regolare, scrivendoli per righe dall'alto verso il basso; in seguito le colonne di T sono *permutate* secondo π. Per cifrare il blocco m_{i+1}, T viene azzerata e il procedimento si ripete.

Il numero di chiavi è teoricamente esponenziale nella lunghezza del messaggio, non essendoci vincoli sulla scelta di r e c.

7.2.4 Cifrari a griglia

Progenitore: **cifrario di Richelieu**.

Il crittogramma può essere celato in un libro qualsiasi e la chiave è data da una scheda perforata e dall'indicazione di una pagina del libro; la decifrazione consiste nel sovrapporre la scheda alla pagina: le lettere visibili attraverso la perforazione costituiscono il messaggio in chiaro.

Variante

La chiave segreta è una griglia quadrata di dimensione $q \times q$, con q pari. $s = \frac{q^2}{4}$ celle della griglia (un quarto del totale) sono trasparenti, le altre opache.

Si scrivono i primi s caratteri del messaggio, nelle posizioni corrispondenti alle celle trasparenti. La griglia viene routata tre volte di 90 gradi in senso orario e si ripete per ogni rotazione l'operazione di scrittura di tre successivi gruppi di s caratteri. Alla fine del processo si fa un *merge* delle tabelle.

- La griglia deve essere costruita in modo che le posizioni corrispondenti alle celle trasparenti non si sovrappongono mai nelle quattro rotazioni.
- Se la lungheza del messaggio è minore di $4s$, le posizioni della pagina P rimaste vuote si riempiono con caratteri scelti a caso.
- Se la lunghezza del messaggio è maggiore di $4s$, il messaggio viene decomposto in blocchi di $4s$ caratteri ciascuno, e ogni blocco è cifrato indipendentemente dagli altri.
- La decifrazione di P è eseguita sovrapponendovi quattro volte la griglia.

Le possibili chiavi sono quante sono le griglie $G = 4^s$. Con $q = 6$ si avrebbero $4^9 \approx 260000$ griglie possibili e per $q = 13, G = d^{36}$ avremmo un numero sufficiente a porre il cifrario al riparo da un attacco esauriente.

Il motivo per cui il numero di griglie è esattamente 4^s è da ricercare nella costruzione stessa della griglia: quando viene costruita una griglia si sceglie la prima cella (da lasciare trasparente) in maniera casuale, di modo che nelle tre successive rotazioni di 90 gradi non ricopra mai lo stesso punto.

7.3 Crittoanalisi statistica

La sicurezza di un cifrario è legata alla dimensione dello spazio delle chiavi: uno spazio delle chiavi vasto (inesplorabile in tempo polinomiale) garantisce protezione da attacchi a forza bruta.

I cifrari storici sono stati violati con un attacco statistico di tipo *cipher text* (il crittoanalista ha a disposizione solo il crittogramma). L'impiego del metodo si fa risalire in Europa alla metà del XIX secolo, quando si scoprì come violare il cifrario di Vigenère, considerato assolutamente sicuro da 300 anni.

Nell'ambito della crittoanalisi statistica, si suppone che il crittoanalista conosca:

- metodo impiegato per la cifratura/decifrazione;
- linguaggio naturale in cui è stato scritto il messaggio;
- si ammette che il messaggio sia sufficientemente lungo per poter rilevare alcuni dati statistici sui caratteri che compongono il crittogramma.

La **crittoanalisi statistica** si basa sulla *frequenza con cui appaiono in media le varie lettere dell'alfabeto*. 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**).

7.3.1 Attacco: sostituzione monoalfabetica

Quando si usa la crittoanalisi per decifrare un crittogramma generato con sostituzione monoalfabetica, si esegue una **prima ipotesi di decifrazione**:

- y nel crittogramma corrisponde a x nel messaggio,
- $\mathit{frequenza}(y) \approx \mathit{frequenza}(x)$.

Perciò, si confrontano le frequenze delle lettere e si provano alcune permutazione tra lettere con frequenze assai prossime: la A e la E compaiono nel 12% dei casi e nel crittogramma c'è una lettera che compare nel 12% circa dei casi, quindi si prova a sostituire la A e la E al posto di quella lettera e si controlla se la sostituzione ha un senso compiuto.

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**. Risolvere il sistema è sempre possibile perché a è invertibile.

Se si tratta di una **cifrario completo** (cioè quando l'alfabeto viene messo in corrispondenza con una permutazione qualunque delle 26! che si possono costruire), allora si associano le lettere in base alle frequenze: le associazioni possono raffinare controllando i digrammi più frequenti. In genere a questo punto il cifrario è completamente svelato; altrimenti si passa ai trigrammi e così via.

7.3.2 Attacco: sostituzione polialfabetica

La decifrazione è più difficile in questo caso: se si costruisce l'istogramma delle frequenze delle lettere del crittogramma, il risultato è generalmente piatto dato che non vi sono corrispondenze dirette fisse che rispettano il linguaggio.

Cifrario di Vigenère

Ogni lettera y del crittogramma dipende da una coppia di lettere $\langle x, k \rangle$ provenienti dal messaggio in chiaro e dalla chiave (si altera la frequenza delle lettere del crittogramma).

La debolezza risiede nella chiave unica e ripetuta più volte: con una chiave di h caratteri è possibile decomporre il crittogramma in varie sottosequenze lunghe h, ciascuna ottenuta per sostituzione monoalfabetica. L'unico problema è scoprire il valore di h: per farlo si sfrutta il fatto che i messaggi contengono quasi sicuramente gruppi di lettere adiacenti ripetuti più volte (trigrammi più frequenti nella lingua, parole a cui il testo si riferisce). Sostanzialmente le apparizioni della stessa sottosequenza allineate con la stessa porzione della chiave sono trasformate nel crittogramma in sottosequenze identiche. Quindi si cercano nel crittogramma coppie di posizioni p_1, p_2 in cui iniziano sottosequenze identiche e a questo punto è molto probabile che $p_2 - p_1$ sia la lunghezza h della chiave o un suo multiplo.

Cifrario di Alberti

È immune da questi attacchi 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.

7.3.3 Attacco: cifrari a trasposizione

Nei crittogrammi generati con questi cifrari, le lettere del crittogramma sono una permutazione delle lettere del messaggio, perciò, eseguire un attacco che si basa sulla statistica andrebbe a generare degli istogrammi che sono identici a quelli generati dagli studi di un determinato linguaggio. Per forzare i cifrari a trasposizione semplice si sfruttano i *q-grammi*: si divide il crittogramma in porzioni di lunghezza h e in ciascuna si cercano i gruppi di q lettere che formano i *q-grammi* più diffusi nel linguaggio (non saranno adiacenti); se un gruppo deriva effettivamente da un *q-gramma*, si scopre parte della permutazione.

7.3.4 Conclusione

La rilevazione delle frequenze delle singole lettere del crittogramma è un potente indizio per discernere tra i vari tipi di cifrario

- nei cifrari a trasposizione l'istogramma delle frequenze coincide approssimativamente con quello proprio del linguaggio;

- nei cifrari a sostituzione monoalfabetica i due istogrammi coincidono a meno di una permutazione delle lettere;
- nei cifrari a sostituzione polialfabetica, l'istogramma del crittogramma è assai più piatto di quello del linguaggio (le frequenze delle lettere variano meno).

7.4 La macchina Enigma

Prima evoluzione verso sistemi automatizzati: si tratta di un'evoluzione elettro-meccanica del cifrario di Alberti. Occupa un ruolo fondamentale nella storia recente (II guerra mondiale) e molti studi dedicati a comprometterne la sicurezza comprendono concetti che hanno portato ai fondamenti dell'informatica teorica.

La chiave è composta dalla configurazione da dare alla macchina e, a seconda della chiave scelta, un tasto premuto sulla tastiera corrisponde all'accensione di una lampadina sulla *lampboard*. Il mittente digita il messaggio e trascrive le corrispondenti accensioni su un foglio da spedire/comunicare al destinatario; il destinatario inserisce la stessa configurazione del mittente, digita i caratteri del mittente e trascrive le accensioni delle lampadine: il messaggio trascritto dalle lampadine è il messaggio in chiaro.

I componenti sono tre rotori di gomma rigida che possono ruotare indipendentemente, il pannello delle lampadine, le barrette dei tasti e un riflettore che è anch'esso un disco come i rotori ma ha un ruolo diverso. Ogni rotore rappresenta una permutazione delle lettere dell'alfabeto ed è composto da due facce, il *pad* e il *pin* su cui sono presenti 26 contatti elettrici; questi contatti elettrici sono messi in comunicazione con i contatti elettrici del rotore ad esso affiancato. Ogni contatto elettrico di un rotore corrisponde ad una lettera, quindi il rotore effettua una vera e propria permutazione.

La forza di Enigma risiede nel fatto che i rotori ruotano; se non fosse stato così allora sarebbe stato un semplice cifrario a sostituzione monoalfabetica con una permutazione semplice dell'alfabeto.

Funzionamento dei rotori

- I rotori non mantenevano la stessa posizione reciproca durante la cifratura.
- Per ogni lettera battuta sulla tastiera il primo rotore avanzava di un passo; dopo 26 passi era tornato sulla posizione iniziale e avanzava di un passo il secondo rotore, dopo 26 avanzava poi il terzo rotore.

La corrispondenza fra caratteri (la chiave) cambiava ad ogni passo in questo modo.

Per un totale di $26 \times 26 \times 26 = 17576$ chiavi diverse.

Debolezze

- I rotori sono immutabili.
- Le 26^3 permutazioni sono sempre le stesse, applicate nello stesso ordine.
- Le permutazioni sono note a tutti i proprietari di una Enigma, mentre Alberti aveva previsto una coppia di dischi per ogni coppia di utenti.

Mitigare le debolezze

- Possibilità di permutare tra loro i tre rotori: le permutazioni diventano $26^3 \times 3! > 10^5$.
- Aggiunta del *plugboard* tra tastiera e primo rotore: consente di scambiare tra loro i caratteri di sei coppie scelte arbitrariamente in ogni trasmissione. In questo modo:
  - Ogni cablaggio è descritto da una sequenza di 12 caratteri (le 6 coppie da scambiare).
  - Combinazioni possibili: $\binom{26}{12} \sim 10^7$.
  - Ogni gruppo di 12 caratteri si può presentare in 12! permutazioni diverse ma non tutte producono effetti diversi sulla cifratura. Difatti, le permutazioni dell'alfabeto che hanno gli stessi scambi di lettere ma in posizioni diverse, non producono chiavi diverse fra loro e queste sono 6!. Inoltre, bisogna dividere per un ulteriore fattore 2^6 perché anche le chiavi prodotte dallo scambio di x con y o di y con x non producono chiavi diverse.

Per concludere: il numero di chiavi è

$$10^5 \cdot \binom{26}{12} \cdot \frac{12!}{(6! \cdot 64)} > 10^{16}$$

un numero più che sufficiente.

II guerra mondiale

- Otto rotori in dotazione, da cui sceglierne tre.
- Aumentarono da sei a dieci le coppie scambiabili nel *plugboard*.

Ogni reparto militare aveva in dotazione un elenco di chiavi giornaliere che conteneva l'**assetto iniziale** della macchina per quel giorno. Con l'assetto iniziale si trasmetteva una nuova **chiave di messaggio** che indicava l'assetto da usare in quella particolare trasmissione.

Capitolo 8

Cifrari perfetti

Sono cifrari che proteggono le informazioni con **sicurezza assoluta** indipendentemente dalle capacità computazionali del crittoanalista. Sono cifrari simmetrici e solo chi possiede la chiave privata del cifrario può decifrare i crittogrammi. Per garantire questa protezione il costo è alto, di conseguenza i cifrari perfetti sono usati solo in casi ristretti (comunicazioni classificate); per le comunicazioni crittografiche di massa si usano cifrari che offrono una *sicurezza di tipo computazionale*: si assume che il crittoanalista abbia accesso a risorse computazionali ragionevoli (polinomiali) e che $P \neq NP$.

Informalmente: un cifrario è perfetto se la sicurezza è garantita qualunque sia l'informazione carpita dal canale.

La pubblicazione ufficiale in letteratura che ha formalizzato il concetto di cifrario perfetto si deve a *Shannon* (1949), un ingegnere e matematico americano. Shannon ha studiato il processo della comunicazione tra mittente e destinatario come se fosse un processo stocastico.

- MSG: spazio dei messaggi
- $CRITTO$: spazio dei crittogrammi
- M: variabile aleatoria che descrive il comportamento del mittente e assume valori in MSG
- C: variabile aleatoria che descrive il processo di comunicazione sul canale e assume valori in $CRITTO$
- $P(M = m)$: probabilità che il mittente voglia spedire il messaggio m
- $P(M = m \mid C = c)$: probabilità condizionata che il messaggio inviato sia m, posto che sul canale transita il crittogramma c

Siamo in una situazione dove il crittoanalista conosce tutto del sistema tranne la chiave; in particolare, conosce la distribuzione di probabilità con cui il mittente invia i messaggi, il cifrario utilizzato e lo spazio delle chiavi (*key*).

Un cifrario è perfetto se

$$\forall\, m \in MSG \text{ e } \forall\, c \in CRITTO\ P(M = m) = P(M = m \mid C = c)$$

Esempio 8.0.1. Supponiamo che

$$\exists\, \bar{m} :\ P(M = \bar{m}) = p > 0$$

$$\exists\, \bar{c} :\ P(M = \bar{m} \mid C = \bar{c}) = 1$$

In questo caso si vuole evidenziare il fatto che se passa $\bar{c}$ sul canale e il crittoanalista è in grado di capire al 100% che si tratta del messaggio $\bar{m}$ cifrato, allora acquisirà conoscenza dall'invio di $\bar{c}$. Ciò accade se e solo se

$$P(M = m) \neq P(M = m \mid C = c)$$

ovvero solo se il cifrario **non** è perfetto.

Si potrebbe anche fare un ulteriore esempio dove

$$\exists\, \bar{\bar{c}} :\ P(M = \bar{m} \mid C = \bar{\bar{c}}) = 0$$

e concluderne che se passa $\bar{\bar{c}}$ canale, il crittoanalista sa sicuramente che il suo corrispettivo non è $\bar{m}$. Nuovamente, il crittoanalista ha appreso nuove informazioni dall'osservazione dei crittogrammi sul canale.

In un cifrario perfetto, la **conoscenza complessiva** del crittoanalista **non cambia** dopo aver osservato un crittogramma in transito: m e c sono del tutto **scorrelati**, nessuna informazione su m può filtrare da c; agli occhi del crittoanalista c appare come una sequenza del tutto casuale.

Teorema 8.0.1 (Shannon)**.** *In un cifrario perfetto, il numero delle chiavi deve essere maggiore o uguale al numero dei messaggi possibili.*

Dimostrazione. Sia N_m il numero dei messaggi possibili, cioè

$$m \in MSG \text{ t.c. } P(M = m) > 0$$

e N_k il numero di chiavi. Supponiamo per assurdo che

$$N_k < N_m$$

e che

$$\exists\, c \in CRITTO \text{ t.c. } P(C = c) > 0$$

Immaginiamo che a c possano corrispondere S messaggi, ovvero tutti quei messaggi m_i che possono essere decifrati a partire da c con la chiave k_i. Si può dire con certezza che

$$S \leq N_k$$

L'ipotesi di partenza era $N_k < N_m$, quindi $S \leq N_k < N_m$. Ma questo significa che $S < N_m$, cioè, il numero di messaggi ottenibili decifrando un crittogramma con ogni chiave è minore del numero dei messaggi totali inviabili con questo sistema di cifratura.

In altre parole

$$\exists\, m \in MSG, P(M = m) > 0 \text{ t.c. } P(M = m \mid C = c) = 0$$

Ma questo vorrebbe dire fornire informazioni al crittoanalista: il cifrario *non* è perfetto $\Rightarrow$ **assurdo** $N_k \geq N_m$.

□

Osservazione Dato che $N_k \geq N_m$, allora le chiavi saranno *molto lunghe*.

8.1 One-Time Pad

Viene introdotto nel 1917 da Mauborgne e Vernam. Usa l'alfabeto binario $\{0, 1\}$, quindi messaggio, crittogramma e chiave sono sequenze di 0 e 1. È stato usato durante la seconda guerra mondiale e negli anni '60 per le comunicazioni tra l'Unione Sovietica e gli Stati Uniti.

La regola di cifratura e decifrazione è pubblica: è lo `XOR` ($\oplus$). Supponiamo che *MSG*, *CRITTO* e *KEY* siano tutte sequenze $\{0, 1\}^n$, ovvero fissiamo un intero n che stabilisce la lunghezza di essi. Quindi, il messaggio $m \in \{0, 1\}^n$ e la chiave $k \in \{0, 1\}^n$, il crittogramma $c = C(m, k)$ lo si ottiene facendo $m \oplus k$ e il messaggio $m = D(c, k)$ lo si ottiene facendo $c \oplus k$. Il motivo per cui si può fare la decifrazione come la cifratura sta nel fatto che

$$m \oplus k = c \Rightarrow c \oplus k = (m \oplus k) \oplus k \Rightarrow m \oplus (k \oplus k) \Rightarrow m$$

Osservazione Nei casi in cui il messaggio m presenti delle regolarità fra i bit, l'operazione $\oplus$ permette di "distruggerle".

La chiave deve essere non riutilizzabile, altrimenti si forniscono informazioni sui messaggi al crittoanalista. Assumiamo di avere due messaggi m' e m''

cifrati con la stessa chiave k, diventano c' e c''. Il crittoanalista potrebbe eseguire l'operazione

$$c' \oplus c'' = (m' \oplus k) \oplus (m'' \oplus k) = m' \oplus m'' \oplus k \oplus k = m' \oplus m''$$

A questo punto quell'operazione potrebbe sembrare innocua ma non lo è affatto: per la proprietà dello $\oplus$ le stringhe uguali danno 0, di conseguenza tutte quelle parti dell'operazione $m' \oplus m''$ che danno 0, indicano al crittoanalista che i messaggi erano uguali. Il cifrario non è perfetto.

One-Time Pad è perfetto

Ipotesi

- Tutti i messaggi hanno lunghezza n: si fa padding se sono più corti, di divide se sono più lunghi.
- Tutte le sequenze di lunghezza n sono messaggi possibili: probabilità molto bassa ma > 0 anche per tutte le sequenze prive di significato, ciò significa che ogni tanto Alice e Bob si inviano messaggi senza senso al solo scopo di confondere il crittoanalista.
- Chiave scelta perfettamente a caso per ogni messaggio.

Sotto queste ipotesi One-Time Pad è un **cifrario perfetto** e **minimale** (impiega un numero minimo di chiavi). Il problema di fondo è la necessità di una chiave segreta lunga quanto il messaggio ogni volta che viene inviato un messaggio: tanto vale scambiarsi il messaggio in segreto.

Dimostrazione.

$$\text{Tesi} : \forall\ m \in MSG \text{ e } \forall\ c \in CRITTO\ P(M = m) = P(M = m \mid C = c)$$

Ricordiamo che

$$P(M = m \mid C = c) = \frac{P(M = m \wedge C = c)}{P(C = c)}$$

Per le proprietà dello $\oplus$ *fissato* un messaggio m, chiavi diverse producono crittogrammi c diversi a partire da questo m; dunque

$$\exists!\ \text{chiave } k \text{ t.c. } m \oplus k = c$$

In questo modo vale che $\forall\ c \in CRITTO$

$$P(C = c) = P(\text{scegliere l'unica chiave } k \text{ t.c. } m \oplus k = c) = \frac{1}{2^n}$$

Questo significa che per ogni messaggio, ottenere un particolare crittogramma è una cosa che dipende solo ed esclusivamente dalla chiave: la probabilità che dato un determinato messaggio m che viene inviato, il suo crittogramma corrispondente sia proprio c è pari alla probabilità stessa di inviare il messaggio (messaggio e crittogramma sono completamente scorrelati). $P(M = m)$ e $P(C = c)$ sono **eventi indipendenti**.

$$\frac{P(M = m \wedge C = c)}{P(C = c)} = \frac{P(M = m) \cdot P(C = c)}{P(C = c)} = P(M = m)$$

Adesso dimostriamo che il cifrario è *minimale*.

Per il teorema di Shannon $N^k \geq N^m$, ma noi sappiamo che se la lunghezza dei messaggi è n allora $N^k = N^m = N^c = 2^n$. Perciò, anche le chiavi sono 2^n e sono il minimo di chiavi possibile: usarne meno non garantisce la sicurezza, dovrei cifrare più messaggi con la stessa chiave, mentre usarne più di 2^n sarebbe inutile, avrei chiavi inutilizzate.

□

Attacchi

Un attacco brute force non ha senso poiché ogni chiave applicata a un crittogramma produce un messaggio che nel sistema OTP esiste: tutte le sequenze di n bit esistevano come entità "messaggio" e avevano una probabilità $p > 0$ di essere inviate, semplicemente quelle che non hanno senso hanno una $p > 0$ ma molto bassa; in pratica, ogni chiave fa ricostruire un messaggio possibile.

Scambiarsi la chiave, però, non è una funzione semplice e ci sono varie alternative: è possibile incontrarsi raramente di persona, scambiarsi una chiave lunghissima e usarla a blocchi (invio un messaggio lungo $n \Rightarrow$ uso n bit della chiave $\Rightarrow$ destinatario riceve e decifra con n bit della chiave). Alternativamente si può usare un generatore pseudo-casuale crittograficamente sicuro facendo sì che sia mittente che destinatario conoscano il tipo di generatore, il seme e i suoi parametri così da generare gli stessi bit. A questo punto vi è un bivio: o si rende pubblico il generatore usato stando però attenti alla periodicità del seme che potrebbe esporre il cifrario ad attacchi basati su chiave ripetuta oppure se ne crea uno ad hoc e lo si mantiene privato. Tipicamente è possibile reperire sul web delle sequenze genomiche sufficientemente casuali: i due estremi della comunicazione possono accordarsi su quale file scegliere e da quale punto iniziare. È un sistema usato e abbastanza sicuro.

Nella dimostrazione che One-Time Pad è un cifrario perfetto, una delle ipotesi fatte era che tutte le sequenze di n bit fossero messaggi possibili. Tuttavia, i messaggi significativi in mezzo a tutti i 2^n messaggi possibili sono

una parte molto più piccola di 2^n; per questo motivo, rimuoviamo l'ipotesi che tutte le sequenze di n bit sono messaggi possibili.
Nota bene: supponiamo che il cifrario non dipende in alcun modo da come cui vengono tradotti i messaggi da linguaggio naturale in binario. I messaggi significativi in lingua inglese, ad esempio, sono circa α^n con $\alpha = 1.1$, perciò $\alpha^n << 2^n$. Avendo annullato l'ipotesi che le sequenze di n bit sono tutte messaggi possibili, possiamo dire che $N_m = \alpha^n$; per il teorema di Shannon

$$N_k \geq N_m \Rightarrow N_k \geq \alpha^n$$

La chiave sarà una sequenza di t bit tale che $2^t \geq \alpha^n$, di modo che l'insieme delle chiavi copra almeno tutto l'insieme dei messaggi.

$$2^t \geq \alpha^n \Rightarrow t \geq \log_2 \alpha^n = n \log_2 \alpha = 0.12 \cdot n$$

Questo significa che il numero di bit necessari per avere una chiave lunga quanto un messaggio significativo sono $t = 0.12 \cdot n$, ovvero è possibile ridurre la lunghezza delle chiavi a $\frac{1}{12}$ dei bit usati precedentemente (quando l'ipotesi che tutte le sequenze di n bit fossero messaggi possibili era ancora valida); tuttavia, i messaggi sono ancora lunghi n bit, perciò, i primi t bit della chiave verranno generati realmente in maniera casuale mentre i successivi $n - t$ bit verranno generati in maniera deterministica.

È opportuno, però, per confondere l'avversario, che coppie messaggio-chiave diverse, producano lo stesso crittogramma. In pratica, decifrando lo stesso crittogramma con svariate delle 2^t chiavi, è possibile ottenere altrettanti messaggi significativi: sarebbe un vantaggio troppo grosso per il crittoanalista se un crittogramma fosse decifrabile con una o poche più chiavi. Per far questo deve valere che

$$\#\text{coppie}\langle m, k\rangle >> \#\text{crittogrammi} \Rightarrow \alpha^n 2^t >> 2^n$$

Da cui si ricava che $t >> 0.88 \cdot n$.

Capitolo 9

DES

DES (Data Encryption Standard) è un **cifrario simmetrico** che ha rappresentato lo standard per le comunicazione di massa (comunicazioni non classificate, "quelle dei cittadini comuni") per moltissimi anni. Come tutti i cifrari progettati per la massa, DES cerca per costruzione di resistere alla crittoanalisi statistica applicando i *principi di Shannon*.

- **Diffusione**: tutti i caratteri del testo in chiaro si devono spargere sul crittogramma dopo la cifratura; in pratica, ogni carattere del crittogramma deve dipendere da tutti i caratteri del testo in chiaro (e anche della chiave).
- **Confusione**: combinare testo in chiaro e chiave in modo complesso così da non permettere al crittoanalista di separarli quando analizza il crittogramma.

Nel 1973 il NBS (National Bureau of Standards)[1] avviò un programma pubblico per proteggere la comunicazione proponendo un bando alle varie accademie per trovare un sistema di cifratura innovativo: la sicurezza doveva basarsi sulla segretezza della chiave, e non sul processo di cifratura e decifrazione (pubblici), e doveva essere certificata da enti di terzi parti; ovviamente l'algoritmo doveva essere efficiente sia in software che in hardware.

Nel 1977 IBM progettò **Lucifer** e NBS lo fece studiare dalla NSA che propose alcune variazioni: chiave da 128 a 56 bit e variazioni nella **S-box**. Questo fece nascere nella IBM il sospetto che la NSA volesse quelle modifiche per poter eseguire brute force in momenti successivi; IBM analizzò comunque per bene le modifiche proposte da NSA e accettò. Quindi nel 1977 DES viene reso pubblicamente disponibile con licenza d'uso gratuita. La certificazione

[1] Oggi si chiama NIST (National Institute of Standards and Technology)

di sicurezza aveva un periodo di rinnovo di 5 anni (ogni 5 anni si controlla se è obsoleto) che si abbassava col passare del tempo: nel 1999 è stato sconsigliato se non per scopi limitati, al suo posto si è iniziato a usare 3DES. 3DES è durato fino al 2005 e dopodiché è stato sconsigliato, mentre dal 2001 entra nello standard l'AES (Advanced Encryption Standard) dopo un bando rimasto aperto dal 1993.

9.1 Funzionamento

La cifratura avviene **a blocchi** di 64 bit e anche la chiave segreta lo è: 56 sono casuali e 8 sono di parità. I bit di parità sono in fondo ad ognuna delle 8 settuple e non sono altro che lo $\oplus$ dei sette bit precedenti.

Il DES consiste di $r = 16$ fasi in cui si ripetono le stesse operazioni e in cui l'output di ogni fase è l'input per la fase successiva e la chiave di ogni fase è diversa: ogni chiave viene selezionata a partire da quella iniziale da 56 bit e la selezione di ogni bit della sottochiave è fatta in modo che ognuno di essi partecipi alla cifratura con pari responsabilità (rapporto di partecipazione: $\frac{14}{16}$).

9.1.1 Struttura del DES

Inizialmente viene eseguita una permutazione del messaggio in chiaro e una trasposizione sulla chiave per scartare i bit di parità; dopodiché il messaggio viene diviso in due parti e insieme alla chiave costituisce l'input per la prima fase. L'output di ogni fase è costituito dal nuovo messaggio elaborato composto nelle sue due metà e da una nuova sottochiave per la fase successiva. Dalla sedicesima fase escono in output solo i due blocchi del crittogramma: questi vengono scambiati e nuovamente concatenati tra loro, successivamente viene eseguita un'altra permutazione.

Per la decifrazione si esegue un procedimento del tutto analogo con le sottochiavi di fase utilizzate nell'ordine inverso.

Funzionamento di una fase

Supponendo di essere nella $(i-1)$-esima fase l'input è dato da

$$\langle S[i-1], D[i-1], k[i-1] \rangle$$

Per ogni $i = 1, 2, \ldots, 16$ si esegue:

$S[i] = D[i-1]$
$D[i] = S[i-1] \oplus f(D[i-1], k[i-1])$

f è una funzione **non lineare**: si tratta della **S-box**.

Tramite lo scambio delle due metà e la combinazione con una funzione non lineare si cerca di realizzare la *diffusione*, mentre con la S-box si cerca di realizzare la *confusione*.

La sottochiave di fase da 56 bit viene divisa in due parti da 28 bit e su entrambe viene eseguito uno shift ciclico verso sinistra di un numero di posizioni che dipende dall'indice della fase: è di una posizione nella fasi 1, 2, 9 o 16 e di due posizioni in tutte le altre. Dopo aver riconcatenato le due chiavi, viene eseguita una permutazione e vengono selezionati 48 bit per eseguire la cifratura di quella fase. Il risultato della concatenazione diventa anche la chiave per la fase successiva.

I 64 bit del messaggio sono divisi in due parti da 32 bit. La parte $D[i-1]$ diventa la parte di $S[i]$ che viene usata nella fase successiva, ma su di essa viene eseguita anche una permutazione e un'espansione tramite duplicazione di bit (crea *diffusione*): i 32 bit iniziale diventano 48. A questo punto, il risultato viene usato per fare lo `XOR` ($\oplus$) con i 48 bit della sottochiave e viene passato alla S-box che esegue un'operazione non lineare e trasforma i 48 bit di entrata in 32 bit che vengono permutati. Infine, eseguo uno `XOR` bit a bit con la parte $S[i-1]$ e si ottiene $D[i]$.

La **S-box** è composta da otto funzioni booleane che prendono in input sei bit e ne restituiscono quattro (48 bit $\rightarrow$ 32 bit). Si prendono i due bit estremi dei sei in input e si usano come `indice di riga`, i restanti quattro bit vengono usati come *indice di colonna*; ogni riga di una S-box è una permutazione dei numeri da 0 a 15 che (non a caso) possono essere rappresentati su quattro bit. A questo punto abbiamo evidenziato una riga e una colonna: l'output della S-box sarà esattamente l'incontro fra le due, ovvero un numero a quattro bit. È molto importante notare che la S-box è una funzione non lineare ovvero una funzione per cui

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

Se non fosse così, DES sarebbe più vulnerabile.

9.2 Attacchi a DES

9.2.1 Attacchi storici

Sono tutti attacchi basati su brute force che si differenziano in due categorie:

- architetture progettate appositamente per attaccare il DES (era abbastanza costoso, circa un milione di dollari per forzare una macchina che usa DES in 35 minuti);

- calcolo distribuito su più macchine (si distribuisce il costo su più persone).

9.2.2 Attacchi esaurienti

$2^{56} - 64$ chiavi deboli = chiavi totali **non deboli** nel DES, rimangono comunque pressoché 2^{56} chiavi; tuttavia esiste un metodo per cui le chiavi di DES non risultano più 2^{56} ma 2^{55}, ovvero vengono dimezzate. Il metodo di attacco si basa sul fatto che

$$C(m, k) = c \text{ e } C(\bar{m}, \bar{k}) = \bar{c}.$$

È un attacco di tipo *chosen plain text*, nel quale il crittoanalista si procura coppie $\langle m, c_1 \rangle, \langle \bar{m}, c_2 \rangle$ e inizia a ispezionare lo spazio delle chiavi. Per ogni chiave k esegue $C(m, k)$:

- se risulta uguale a c_1, allora k è probabilmente la chiave;
- se risulta uguale a $\bar{c}_2$, allora $\bar{k}$ è probabilmente la chiave;
- se nessuno dei precedenti casi, allora né k né $\bar{k}$ sono chiave.

Ma perché si può fare questa affermazione? Eseguiamo $C(m, k)$ e otteniamo $\bar{c}_2$: per la proprietà di DES $C(\bar{m}, \bar{k}) = \bar{c}_2 = c_2$ e ricordiamo che il crittoanalista conosceva la coppia $\langle \bar{m}, c_2 \rangle$. Provando una qualsiasi chiave k e la sua verifica, è possibile scoprire immediatamente se k o $\bar{k}$ sono chiave; di conseguenza, in un colpo solo *si possono verificare due chiavi o se ne possono scartare due*: si **dimezza lo spazio delle chiavi**.

9.2.3 Attacchi con crittoanalisi differenziale (Biham e Shamir, 1990)

Sono sempre attacchi *chosen plain text* nei quali si prendono 2^{47} coppie $\langle m, c \rangle$ scelti dal crittoanalista. L'idea molto semplificata è di scegliere meticolosamente le coppie $\langle m, c \rangle$ e vedere come le differenze fra i vari messaggi si ripercuotono sui crittogrammi al solo scopo di assegnare delle probabilità alle varie chiavi che potrebbero essere state usate per avere quelle determinate ripercussioni: alla fine si ottengono chiavi molto più probabili di altre. Il costo di un attacco simile è di $2^{55.1}$ ed il motivo per cui è così alto è dovuto al numero di fasi del DES (16). Gli sviluppatori erano già a conoscenza della crittoanalisi differenziale e si sono protetti proprio col numero di fasi.

9.2.4 Attacchi con crittoanalisi lineare (Matsui, 1993)

Sono attacchi che prendono la funzione di cifratura, la approssimano con una funzione lineare per stimare qualche bit della chiave e cercano di completare il resto della chiave con un attacco brute force. Necessita di 2^{43} coppie $\langle m, c \rangle$ ed è un attacco *known plain text*. L'attacco è più efficiente di 2^{55} e quindi più efficiente di un brute force $\rightarrow$ DES è stato ufficialmente forzato.

Capitolo 10

AES

DES necessitava di un irrobustimento data la vulnerabilità alla crittoanalisi differenziale e lineare. La prima proposta fu quella di scegliere in maniera indipendente le 16 sottochiavi di fase; questo ampliò il numero di bit delle chiavi da 56 a $16 \cdot 48 = 768$ bit illudendo di avere uno spazio delle chiavi pari a 2^{768}: con la crittoanalisi differenziale fu dimostrato che la cardinalità dello spazio delle chiavi era diventata 2^{61} contro i 2^{56} precedenti.

La seconda proposta fu quella della **cifratura multipla**: l'idea è di comporre il DES con se stesso. Scegliendo due chiavi k_1 e k_2 è stato dimostrato che

$$C(C(m, k_1), k_2) \neq C(m, k_3) \quad \forall\ m, k_3$$

In pratica, applicare DES n volte è diverso dall'applicarlo una volta sola, quindi c'è un aumento di sicurezza. Si potrebbe pensare che usando due chiavi di lunghezza 56 bit, lo spazio delle chiavi sia di 2^{112}, ma non è così: non sono tutti di sicurezza. In realtà, l'attacco *Meet in the Middle* dimostra che la sicurezza è pari a quella di una chiave lunga 57 bit.

Meet in the Middle

Si prenda

$$c = C(C(m, k_1), k_2)$$

con k_1, k_2 chiavi di 56 bit e che

$$D(c, k_2) = C(m, k_1)$$

Il crittoanalista prende una coppia $\langle m, c \rangle$ e $\forall\ k_1$ calcola e salva $C(m, k_1)$ in una lista; mentre $\forall\ k_2$ si calcola $D(c, k_2)$ e lo cerca nella suddetta lista.

Osservazione Per costruzione dell'attacco, deve esistere certamente almeno una corrispondenza fra $D(c, k_2)$ e $C(m, k_1)$.

Il costo dell'attacco è $O(2^{57})$: il crittoanalista deve aver calcolato e messo in una lista $C(m, k_1)$ per tutte le chiavi k_1, cioè 2^{56}; successivamente deve calcolare al più 2^{56} volte $D(c, k_2)$. Quindi:

$$2^{56} + O(2^{56}) = O(2 \cdot 2^{56}) = O(2^{57})$$

che è ben diverso da 2^{112}.

10.1 3DES

Alla fine è stato adottato il 3DES che viene indicato con

- 2TDEA: Triple Data Encryption Algorithm con due chiavi;
- 3TDEA: Triple Data Encryption Algorithm con tre chiavi;

2TDEA

$c = C(D(C(m, k_1), k_2), k_1)$ dove k_1 e k_2 sono chiavi di 56 bit, tra loro indipendenti.

Si osservi che se $k_1 = k_2$, allora 2TDEA equivale a DES con singola cifratura ed è stato fatto per mantenere la compatibilità con quei sistemi che erano abilitati a usare solo un singolo DES. Non è più robusto di una tripla cifratura.

Il livello di sicurezza in bit è 112; ci vogliono 2^{112} operazioni di cifratura e decifrazione: ha un costo equivalente al forza bruta, quindi il triplo DES con due chiavi non è vulnerabile agli attacchi *meet in the middle*.

3TDEA

$c = C(D(C(m, k_1), k_2), k_3)$ dove k_1 e k_2 sono chiavi di 56 bit. Ci si aspetterebbe che la sicurezza sia di $56 \cdot 3 = 168$ bit, ma in realtà è vulnerabile a *meet in the middle* e la sicurezza risulta essere di 2^{112}.

Dimostrazione. $m = D(C(D(c, k_3), k_2), k_1)$. Per costruzione si ricava che $C(m, k_1) = C(D(c, k_3), k_2)$ a questo punto è possibile:

- enumerare le chiavi k_1 di 56 bit e salvare $C(m, k_1)$ in una lista;
- $\forall\ k_2, k_3$ calcolare $C(D(c, k_3), k_2)$ e cercarlo nella lista.

Il costo complessivo totale è di $2^{56} + 2^{112}$, cioè il costo di enumerare tutte le chiavi k_1 sommato al costo di enumerare tutte le coppie $\langle k_2, k_3 \rangle$. □

10.2 AES

Nel Giugno 1998 fu proposto dal NIST un bando per rimpiazzare il DES e furono presentate ventuno proposte. I parametri di cui tener conto erano la sicurezza, il costo di realizzazione (doveva essere economico in hardware ed efficiente con le operazioni di cifratura e decifrazione, con meno spazio occupato possibile e licenza gratuita), la correttezza, la casualità e le caratteristiche algoritmiche (flessibile, portabile, chiavi con diversi numeri di bit [128-256]). Nell'Agosto '98 rimasero quindici cifrari validi fino ad Aprile '99 quando diminuirono a cinque e nell'Ottobre 2000 fu scelto l'AES divenendo il nuovo standard nel 2001.

È un cifrario molto efficiente, leggero, flessibile e permette di usare chiavi da 128, 192 o 256 bit e si lavora sempre a gruppi di 32 bit (estendibili). Noi vedremo la variante con 128 bit di blocco e 128 bit di chiave. Anche AES è un cifrario a fasi e il numero di esse dipende dal numero di bit di chiave: 128 bit sono 10 fasi, 192 bit sono 12 fasi e 256 bit sono 14 fasi.

Selezione delle sottochiavi di fase

Si parte da una chiave lunga 128 bit e a partire da questa si generano deterministicamente le sottochiavi di fase. Per generare queste sottochiavi si inserisce la chiave della fase precedente in una matrice $k = 4 \times 4$ byte, caricandola per colonna. Ogni sottofase riceve in input una nuova matrice k calcolata a partire dalla matrice precedente. L'innovazione è l'impiego della S-box, non solo durante il processo di cifratura, ma anche per generare le chiavi delle fasi.

Inizialmente abbiamo in input quattro parole di chiave:

$$w(0), w(1), w(2), w(3)$$

Poi $\forall\ t \geq 4$

$$W(t) = \begin{cases} w(t-1) \oplus w(t-4) & \text{se } 4 \nmid t \\ T(w(t-1)) \oplus w(t-4) & \text{se } 4 \mid t \end{cases}$$

Quindi l'S-box interviene solo nel caso in cui t sia multiplo di 4.

La chiave dell'i-esima fase, con $1 \leq i \leq 10$ è

$$w(4i), w(4i+1), w(4i+2), w(4i+3)$$

Cifratura

Si hanno blocchi di 128 bit che vengono caricati (nuovamente) per colonna in una matrice 4×4 byte.

$$B = \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix} \qquad b_{ij} \in \{0,1\}^8$$

Prima di procedere con la cifratura a 10 fasi, si esegue una **trasformazione iniziale**.

$$B \to B \oplus k \qquad k : \text{matrice della chiave}$$

Adesso possono iniziare le 10 fasi, le quali sono composte da quattro operazioni ciascuna:

1. **Substitute bytes**: ogni byte di B è trasformato usando una S-box, perciò $b_{ij} \to S\text{-}box(b_{ij})$.
2. **Shift rows**: shift ciclico sulle righe.
3. **Mix columns** (non si applica nella fase 10).
4. **Add round key**: aggiunta della chiave a B.

Le prime tre operazioni servono per garantire i principi di Shannon, mentre l'ultima è semplicemente la cifratura con chiave. Alla fine delle 10 fasi e dell'uso di queste operazioni in ognuna di esse, il blocco B diventa il crittogramma.

Decifrazione

Simile alla cifratura, ma avviene al contrario e ci sono alcune differenze nelle quattro operazioni.

10.2.1 S-box

Substitute byte

È una matrice 16×16 di interi $\in [0, 255]$ e contiene una permutazione di essi (ogni intero compare al massimo una volta). Gli interi sono rappresentabili su un byte ($2^8 = 256$).

$$b_{ij} \to S\text{-}box(b_{ij})$$

La S-box prende in input un intero fra 0 e 255 e lo trasforma nel seguente modo: i primi 4 bit del numero da trasformare identificano la riga della matrice 16×16 mentre i secondi 4 bit rappresentano la colonna della medesima

matrice. Il numero di uscita della S-box è il numero che si trova nell'intersezione riga-colonna. Ciò che si ottiene è l'inverso moltiplicativo del byte in input calcolato rispetto al campo di Galois di cardinalità 2^8 (GF(2^8)[1]).

La composizione algebrica di questa S-box è la stessa usata per calcolare i byte delle sottochiavi di fase.

Shift rows

$$\begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix} \rightarrow \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{11} & b_{12} & b_{13} & b_{10} \\ b_{22} & b_{23} & b_{20} & b_{21} \\ b_{33} & b_{30} & b_{31} & b_{32} \end{bmatrix}$$

Molto semplicemente, quest'operazione non fa altro che applicare *diffusione* sul blocco.

Mix columns

Viene fissata una matrice $M = 4 \times 4$ byte.

$$B_j \rightarrow M \times B_j \qquad 0 \leq j \leq 3$$

il nuovo b_{ij} diventa un valore che dipende da tutti i byte della colonna.

Add round key

Molto semplice:

$$b_{ij} \rightarrow b_{ij} \oplus k_{ij}$$

[1]La somma è *mod* 2, mentre la moltiplicazione è *mod* 2^8

Capitolo 11

Cifratura a blocchi e a chiave pubblica

11.1 Cifratura a blocchi

Cifrari come l'AES e il DES lavorano a blocchi: se si ha un messaggio di n bit, questo **viene diviso e cifrato in vari blocchi** sempre con la stessa chiave. Si noti che un messaggio ha la seguente forma:

$$m = m_1 m_2 m_3 \dots m_l \text{ t.c. } |m_i| = 128 \text{ bit}$$

Qualora non valga che $|m_l| < 128$ bit allora si aggiunge una concatenazione t.c. $|m_l 100 \dots 0| = 128$ bit. Se invece $|m_l| = 128$ bit allora si aggiunge un blocco terminatore $|10 \dots 0| = 128$. *Per evitare che messaggi diversi diano crittogrammi uguali*, al messaggio viene concatenata una stringa iniziale c_0 che potrebbe rappresentare anche il timestamp in cui viene cifrato il messaggio.

La **cifratura** è semplice:

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

si prende l'i-esimo blocco del messaggio da cifrare, si esegue lo $\oplus$ col crittogramma c_{i-1} calcolato per il blocco precedente e infine si cifra il risultato ottenuto con la chiave k. Sostanzialmente per cifrare il blocco di messaggio $(i+1)$-esimo è necessario aver già cifrato il blocco di messaggio i-esimo.

La **decifrazione** è anch'essa semplice: dato che la doppia applicazione di una stringa con lo $\oplus$ ha l'effetto di annullare l'effetto stesso dello $\oplus$, allora per decifrare c_i si esegue

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

Ci sono proprietà molto interessanti nella cifratura a blocchi:

- La decifrazione può essere fatta *in parallelo* perché il blocco c_{i-1} è già noto nel momento in cui bisogna decifrare il blocco c_i.
- La cifratura a blocchi prevede che il crittogramma c_i influenzi solo se stesso e c_{i+1}. Questo significa che se alcuni bit di c_i sono trasmessi male, la decifratura stessa di c_i non influenzerà tutto il messaggio m ma solo i blocchi m_i e m_{i+1}.

11.2 Crittografia a chiave pubblica

Per tanto tempo, il problema principale della crittografia è stato lo **scambio della chiave segreta**. Questo problema è rimasto tale fino al 1976, quando Diffie e Hellman hanno proposto due metodi per generare e scambiare una chiave segreta su un canale insicuro senza che le due parti si debbano incontrare precedentemente. Il primo metodo è un algoritmo chiamato **protocollo DH** ed è ancora usato nella crittografia su Internet; il secondo è la **crittografia a chiave pubblica** che non prevede affatto alcuno scambio di chiave. In quest'ultimo metodo sostanzialmente le due parti cifrano il messaggio con una chiave pubblica e lo decifrano rispettivamente con una chiave privata.

Nei cifrari a chiave privata (o simmetrici) si ha una chiave segreta per ogni coppia di utenti che vuole parlare, facendo crescere il numero di chiavi con un fattore esponenziale.

Nei cifrari a chiave pubblica o cifrari asimmetrici, invece, tutti possono inviare messaggi cifrati e solo il ricevente può decifrarli. Le operazioni di cifratura e decifrazione sono pubbliche e utilizzano due chiavi diverse:

- k_{pub} per cifrare: è pubblica, nota a tutti.
- k_{priv} per decifrare: è privata, nota solo al destinatario.

In questo caso, ogni utente possiede una coppia $\langle k_{pub}, k_{priv} \rangle$: la chiave pubblica è nota a chiunque e qualsiasi persona interessata a inviare un messaggio all'utente che possiede k_{pub} come chiave pubblica, può e deve cifrare il proprio messaggio esattamente con k_{pub}. Il destinatario provvederà a decifrare con la sua chiave privata k_{priv} nota solo a lui. In questo modo il numero di chiavi è $2n$ se si hanno n utenti.

Cifratura e decifrazione in un cifrario a chiave pubblica

Cifratura: $c = C(m, k_{pub})$.
Decifrazione: $m = D(c, k_{priv})$.

I requisiti perché un cifrario a chiave pubblica funzioni sono:

1. Correttezza: per ogni possibile messaggio $m = D(C(m, k_{pub}), k_{priv})$.
2. Efficiente e sicuro: tutto ciò che è lecito, legale, deve richiedere tempo polinomiale, mentre tutto ciò che è "illegale" deve richiedere tempo esponenziale. Quindi:
   - la **coppia di chiavi è facile da generare** e deve risultare pressoché impossibile che due utenti scelgano la stessa chiave (*generazione casuale delle chiavi*);
   - dati m e k_{pub}, **è facile calcolare il crittogramma** $c = C(m, k_{pub})$ (*adottabilità del sistema*);
   - dati c e k_{priv}, **è facile calcolare il messaggio in chiaro** $m = D(c, k_{priv})$ (*adottabilità del sistema*);
   - pur conoscendo il crittogramma c, k_{pub}, C e D la **decifrazione** di c deve essere **difficile per il crittoanalista** (*sicurezza del cifrario*).

Al fine di poter soddisfare questi requisiti, per la funzione di cifratura C si deve ricorrere a una funzione **one-way trapdoor**, cioè calcolare $c = C(m, k_{pub})$ è *computazionalmente facile*, ma calcolare $m = D(c, k_{priv})$ è *computazionalmente difficile* se non si conosce la trapdoor (k_{priv}). Tuttavia, Diffie e Hellman non riuscirono mai a trovare una funzione one-way trapdoor.

11.2.1 RSA

Rivest, Adleman e Shamir proposero un sistema a chiave pubblica riuscendo a trovare una funzione one-way trapdoor. RSA si basa sulla moltiplicazione di due numeri primi p e q:

- Calcolare $n = p \cdot q$ è facile. Una moltiplicazione richiede sempre tempo polinomiale.
- Calcolare p e q conoscendo n è difficile a meno che non si conosca uno dei due fattori. In pratica, fattorizzare n senza conoscere p o q richiede tempo esponenziale.

RSA **utilizza algebra modulare** (*mod* n).

11.2.2 Algebra modulare

Usata in molti algoritmi crittografici per

- ridurre lo spazio dei numeri su cui si opera e quindi aumentare la velocità di calcolo;
- rendere difficili problemi computazionali che sono semplici (o anche banali) nell'algebra non modulare.

In algebra modulare le funzioni tendono a comportarsi in modo "*imprevedibile*". Si prenda, per esempio, la funzione 2^x. Il suo comportamento nell'algebra ordinaria è monotono crescente:

x	1	2	3	4	5	6	7	8	9	10	11	12
2^x	2	4	8	16	32	64	128	256	512	1024	2048	4096

Se invece si prende $2^x \ mod\ 13$, il comportamento è:

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

Sale, scende, sale... ha un andamento caotico: perde struttura rispetto alla precedente e non dà alcun suggerimento.

Caratteristiche

Preso $n \in \mathbb{N}$, intero positivo

$$\mathbb{Z}_n = \{0, 1, 2. \ldots, n-1\} \qquad \mathbb{Z}_n^* \subseteq \mathbb{Z}_n$$

è l'insieme degli elementi di $\mathbb{Z}_n$ co-primi con n. Se n è primo, $\mathbb{Z}_n^* = \mathbb{Z}_n$. Se n non è primo, calcolare $\mathbb{Z}_n^*$ è computazionalmente difficile: richiede tempo proporzionale al valore di n (per confrontare n con gli elementi di $\mathbb{Z}_n$) quindi esponenziale nella lunghezza della sua rappresentazione.

Dati due interi $a, b \geq 0$ e $n \geq 0$, a è congruo a b modulo n

$$a \equiv b \ mod\ n$$

se e solo se esiste k intero per cui

$$a = b + kn$$

Nelle relazioni di congruenza la notazione $mod\ n$ si riferisce all'intera relazione, nelle relazioni di uguaglianza la stessa notazione si riferisce solo al membro dove appare: $5 \equiv 8 \ mod\ 3$, ma $5 \neq 8 \ mod\ 3$

Proprietà

$$(a+b)\ mod\ m = (a\ mod\ m + b\ mod\ m)\ mod\ m$$
$$(a-b)\ mod\ m = (a\ mod\ m - b\ mod\ m)\ mod\ m$$
$$(a\times b)\ mod\ m = (a\ mod\ m \times b\ mod\ m)\ mod\ m$$
$$a^{r\times s}\ mod\ m = (a^r\ mod\ m)^s\ mod\ m \quad (r, s \text{ interi positivi})$$

Funzione di Eulero

Il numero di interi minori di n e co-primi con esso

$$\phi(n) = |\mathbb{Z}_n^*|$$

Se n è primo $\Rightarrow \phi(n) = n - 1$.

Teorema 11.2.1.

$$n \text{ composto} \Rightarrow \phi(n) = n\left(1-\frac{1}{p_1}\right)\ldots\left(1-\frac{1}{p_k}\right)$$

$p_1, \ldots, p_k$ fattori primi di n, presi senza molteplicità.

Teorema 11.2.2. *n prodotto di due primi (semi-primo)*

$$n = p\cdot q \Rightarrow \phi(n) = (p-1)(q-1)$$

Teorema 11.2.3 (Eulero)**.** *Per $n > 1$ e per ogni a primo con n*

$$a^{\phi(n)} \equiv 1\ mod\ n$$

Teorema 11.2.4 (Fermat)**.** *Per n primo e per ogni $a \in \mathbb{Z}_n^*$*

$$a^{n-1} \equiv 1\ mod\ n$$

Conseguenze

Per qualunque a primo con n

$$a \times a^{\phi(n)-1} \equiv 1\ mod\ n \quad \text{(Teorema di Eulero)}$$
$$a \times a^{-1} \equiv 1\ mod\ n \quad \text{(definizione di inverso)}$$

quindi,

$$a^{-1} = a^{\phi(n)-1}\ mod\ n$$

L'inverso a^{-1} di a modulo n si può dunque calcolare per esponenziazione di a se si conosce $\phi(n)$. In generale, nell'algebra modulare l'esistenza dell'inverso non è garantita perché a^{-1} deve essere intero.

Teorema 11.2.5. *L'equazione* $ax \equiv b \ mod \ n$ *ammette soluzione se e solo se* $\text{MCD}(a, n)$ *divide* b*. In questo caso si hanno esattamente* $\text{MCD}(a, n)$ *soluzioni distinte.*

Corollario 11.2.1. *L'equazione* $ax \equiv b \ mod \ n$ *ammette un'unica soluzione se e solo se* a *e* n *sono co-primi (*$\text{MCD}(a, n) = 1$*)* $\Leftrightarrow$ *esiste l'inverso* a^{-1} *di* a*.*

Se nel corollario si sostituisce 1 con b, si ottiene $ax \equiv 1 \ mod \ n$. Ne consegue che ammette esattamente una soluzione (l'inverso di a) se e solo se a e n sono primi tra loro. L'inverso si può calcolare come

$$a^{-1} = a^{\phi(n)-1} \ mod \ n$$

ma occorre conoscere $\phi(n)$, cioè fattorizzare n: è un problema "difficile".

Algoritmo di Euclide Esteso

L'algoritmo di Euclide per il calcolo del MCD si può estendere per risolvere l'equazione in due incognite $ax + by = \text{MCD}(a, b)$.

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

Osservazioni:

- La funzione `Extended_Euclid` restituisce una delle triple di valori

  $$\langle \text{MCD}(a, b), x, y \rangle$$

  con x, y tali che $ax + by = \text{MCD}(a, b)$. Quindi $d = \text{MCD}(a, b)$.

- complessità logaritmica nel valore di a e b, quindi *polinomiale* nella dimensione dell'input.

L'algoritmo di Euclide esteso si può applicare al **calcolo dell'inverso**.

$ax \equiv 1 \ mod \ b \Leftrightarrow ax = bz + 1$ per un opportuno valore di z se e solo se $ax + by = \text{MCD}(a, b)$, dove $y = -z$ e $\text{MCD}(a, b) = 1$.

Generatori

$a \in \mathbb{Z}_n^*$ è un **generatore** di $\mathbb{Z}_n^*$ se la funzione

$$a^k \ mod \ n \qquad 1 \leq k \leq \phi(n)$$

genera tutti e soli gli elementi di $\mathbb{Z}_n^*$. Produce come risultati tutti gli elementi di $\mathbb{Z}_n^*$, ma in un ordine difficile da prevedere.

Teorema 11.2.6 (Eulero)**.** *$a^{\phi(n)} \equiv 1 \ mod \ n \Rightarrow 1 \in \mathbb{Z}_n^*$ è generato per $k = \phi(n)$. Per ogni generatore*

$$a^k \not\equiv 1 \ mod \ n \qquad 1 \leq k < \phi(n)$$

Teorema 11.2.7. *Se n è un numero primo, $\mathbb{Z}_n^*$ ha almeno un generatore.*

- Per n primo, non tutti gli elementi di $\mathbb{Z}_n^*$ sono suoi generatori (1 non è mai generatore e altri elementi non possono esserlo).
- Per n primo, i generatori di $\mathbb{Z}_n^*$ sono in totale $\phi(n-1)$.

Problemi sui generatori rilevanti in crittografia

Risolvere nell'incognita x l'equazione $a^x = b \ mod \ n$, con n primo.

L'equazione ammette una soluzione per ogni valore di b se e solo se a è un generatore di $\mathbb{Z}_n^*$. Tuttavia non è noto a priori in che ordine sono generati gli elementi di $\mathbb{Z}_n^*$, quindi non è noto per quale valore di x si genera $b \ mod \ n$. Un esame diretto della successione richiede tempo esponenziale nella dimensione di n: non è noto un algoritmo polinomiale di soluzione.

11.2.3 Funzioni one-way trap-door

Esistono funzioni matematiche che sembrano possedere i requisiti richiesti: il loro calcolo risulta incondizionatamente semplice e la loro inversione semplice se si dispone di un'informazione aggiuntiva sui dati (cioè una chiave privata). Senza questa informazione, l'inversione richiede la soluzione di un problema NP-hard, o comunque di un problema noto per cui non si conosce un algoritmo polinomiale.

Fattorizzazione

Calcolare $n = p \times q$ è facile e richiede tempo quadratico nella lunghezza della loro rappresentazione. Invertire la funzione per trovare p e q a partire da n (univocamente possibile solo se p e q sono primi) richiede tempo

(sub)esponenziale. Per quanto noto fino a oggi, non è mai stato dimostrato che il problema è NP hard ma non è nemmeno mai stato dimostrato che il problema ammette un algoritmo risolutivo polinomiale (cosa da non escludere). La trap door in questo caso è uno qualsiasi dei due fattori p e q.

Calcolo della radice in modulo

Calcolare $y = x^z \ mod \ s$ con x, z, s interi, richiede tempo polinomiale e si può fare con l'algoritmo delle esponenziazioni successive eseguendo $\Theta(\log_2 z)$ moltiplicazioni. Se s non è primo, invertire la funzione e calcolare $x = y^{\frac{1}{z}} \ mod \ s$ richiede tempo esponenziale per quanto noto ad oggi. La trap door la vedremo successivamente.

Calcolo del logaritmo discreto

Calcolare la potenza $y = x^z \ mod \ s$ è facile. Invertire rispetto a z, cioè trovare z t.c. $y = x^z \ mod \ s$ dati x, y e s è computazionalmente difficile. Tutti gli algoritmi noti hanno la stessa complessità della fattorizzazione.

11.2.4 Vantaggi e svantaggi

Vantaggi

- Se ho n utenti, il numero di chiavi del sistema sono $2n$ anziché $\frac{n(n-1)}{2}$.
- Non è richiesto alcuno scambio di chiavi.

Svantaggi

- Molto più lenti dei cifrari simmetrici.
- Sono esposti per natura ad attacchi di tipo *chosen plain-text*.

Attacchi chosen plain-text

Un crittoanalista può crearsi svariati crittogrammi $c = C(m, k_{pub})$ di un determinato destinatario e successivamente può mettersi in ascolto sul canale verso il quale sono diretti i crittogrammi per quel determinato destinatario. A questo punto il crittoanalista può confrontare i crittogrammi che passano sul canale con quelli che si era preparato in precedenza e, conoscendone il testo in chiaro, se ne trova due uguali ne conosce automaticamente la decifrazione. Qualora non trovasse alcuni dei suoi crittogrammi uguali a quelli passati sul canale, saprebbe comunque che il messaggio in chiaro è diverso da quelli da lui preparati.

11.3 Cifrari ibridi

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.

La trasmissione dei messaggi lunghi avviene ad alta velocità, mentre lo scambio delle chiavi segrete è lento (sono composte al massimo da qualche decina di byte). L'attacco *chosen plain-text* è risolto 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*.

Capitolo 12

RSA

Generazione della chiave

Il destinatario è colui che deve creare la chiave, quindi

1. sceglie p e q numeri primi, molto grandi (migliaia di bit) tale che $n = p \times q$ abbia almeno 2048 cifre binarie (ad oggi si consigliano 3072 bit): si generano due sequenze casuali di mille bit e poi se ne testa la primalità con Miller-Rabin, questo garantisce una generazione polinomiale;

2. calcola $n = p \times q$ e la funzione di Eulero (che rappresenta il numero di interi minori di n e coprimi con n) $\phi(n) = (p-1) \times (q-1)$: calcolare $p \times q$ e la funzione di Eulero risulta molto semplice (se si conosce la scomposizione in fattori di n), perciò, dato che per il destinatario ciò è valido, si ha un altro passo di tempo polinomiale;

3. sceglie $e < \phi(n)$ t.c. MCD$(e, \phi(n)) = 1$: nuovamente, si ha l'algoritmo di Euclide per il controllo del MCD che permette di avere tempo polinomiale;

4. calcola $d = e^{-1} \ mod \ \phi(n)$ (la condizione precedente garantisce che d esista e sia unico): il calcolo dell'inverso in modulo si può fare con l'algoritmo di Euclide esteso che nuovamente consente di avere tempo polinomiale.

Il destinatario ha terminato le operazioni da svolgere ed ha finalmente ottenuto le sue chiavi: $k_{pub} = \langle e, n \rangle$ e $k_{priv} = \langle d \rangle$.

Messaggio

Si tratta di una sequenza binaria trattata come un intero. Un qualsiasi messaggio m deve rispettare la condizione che $|m| < n$, altrimenti m e $m \ mod \ n$

finirebbero nello stesso crittogramma e ci sarebbe ambiguità durante la decifrazione: se $m \geq n$, si divide in blocchi di

$$|b| = \lfloor (\log_2 n) \rfloor \text{ bit}$$

cifrati indipendentemente.

Si noti che nella pratica si fissa un limite comune per la dimensione dei blocchi b: $m < 2^b < n$. Questo garantisce anche una sorta di sicurezza poiché più n è piccolo e più il cifrario è poco sicuro, quindi fissare un limite inferiore alla dimensione dei blocchi (b) ci permette di dire di conseguenza che n avrà un valore alto.

Cifratura

$$c = C(m, k_{pub}) = m^e \ mod \ n$$

Data la presenza del modulo, $c < n$. Si noti inoltre che la presenza del modulo è cruciale: senza il modulo, c sarebbe semplicemente $\sqrt[e]{m}$ ovvero un problema molto facile. Col modulo diventa un problema del quale non è noto alcun algoritmo polinomiale!

Decifrazione

$$m = D(c, k_{priv}) = c^d \ mod \ n$$

Sia cifratura che decifrazione possono essere eseguite in tempo polinomiale grazie all'algoritmo delle quadrature successive.

12.1 Correttezza

La correttezza prevede che

$$D(C(m, k_{pub}), k_{priv}) = m$$

In questo caso è necessario dimostrare

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

Teorema 12.1.1 (Correttezza)**.** $\forall \ m < n, \ m^{ed} \ mod \ n = m$

Dimostrazione. Se p e q non dividono $m \Rightarrow \text{MCD}(m, n) = 1$ (m e n sono co-primi). Se ne ricava che

1. $m^{\phi(n)} \equiv 1 \ mod \ n$ (Teorema di Eulero)

2. $e \cdot d \equiv 1 \ mod \ \phi(n) \Rightarrow e \cdot d = 1 + r\phi(n) \quad r \in \mathbb{N}$ (def. di inverso)

Per quanto detto

$$m^{ed} \ mod \ n = m^{1+r\phi(n)} \ mod \ n = m \left(m^{\phi(n)}\right)^r \ mod \ n = m \cdot 1^r \ mod \ n = m$$

Se m e n non sono coprimi: supponiamo che $p \mid m$ e $q \nmid m$. Dato che $p \mid m$, allora $m \equiv 0 \ mod \ p$ e quindi $\forall \ r \in \mathbb{N}, \ m^r \equiv 0 \ mod \ p$ da cui $m^r - m \equiv 0 \ mod \ p$. In particolare è possibile scegliere $r = e \cdot d$, quindi $m^{ed} - m \equiv 0 \ mod \ p$. Dato che $q \nmid m$ allora $\text{MCD}(q, m) = 1$ e quindi $m^{\phi(q)} \equiv 0 \ mod \ q$.

$$m^{ed} \ mod \ q = m^{1+r\phi(n)} \ mod \ q = m \cdot m^{r(p-1)(q-1)} \ mod \ q =$$

$$= m \left(m^{(q-1)}\right)^{r(p-1)} \ mod \ q = m \left(m^{\phi(q)}\right)^{r(p-1)} \ mod \ q =$$

$$= m(1)^{r(p-1)} \ mod \ q = m \ mod \ q$$

Perciò

$$m^{ed} \equiv m \ mod \ q \Rightarrow m^{ed} - m \equiv 0 \ mod \ q$$

Siccome $m^{ed} - m$ è divisibile per p e per q, allora è divisibile anche per $n = p \times q$.

$$m^{ed} - m \equiv 0 \ mod \ n \Rightarrow m^{ed} \equiv m \ mod \ n \Rightarrow m^{ed} \ mod \ n = m \ mod \ n = m$$

Si potrebbe considerare anche il caso in cui $p, q \mid m$, ma in realtà non si verifica perché $m < n$. □

12.2 Sicurezza

La sicurezza è basata sulla difficoltà di fattorizzare un numero arbitrario molto grande.

Fattorizzare $\Rightarrow$ forzare RSA: è sicuramente valido.
Forzare RSA $\Leftarrow$ fattorizzare: è plausibile ma non certo.

Calcolare la radice in modulo per trovare $m = \sqrt[e]{c} \ mod \ n$ (che equivale a decifrare un crittogramma c) è difficile almeno quanto fattorizzare.

Calcolare $\phi(n)$ direttamente da n (e ciò basterebbe per ricavare la chiave privata) è difficile almeno quanto fattorizzare.

È possibile dimostrare che fattorizzare n e calcolare $\phi(n)$ sono computazionalmente equivalenti (uno si trasforma nell'altro in tempo polinomiale):

Fattorizzazione di $n \Rightarrow$ calcolo di $\phi(n)$:

$$n = p \times q \rightarrow \phi(n) = (p-1)(q-1)$$

in tempo polinomiale.

n e $\phi(n) \Rightarrow$ trovo p e q in tempo polinomiale.

$$\phi(n) = (p-1)(q-1) = pq - (p+q) + 1 = n - (p+q) + 1$$

$$\Rightarrow x_1 = p + q = n - \phi(n) + 1$$
$$(p-q)^2 = (p+q)^2 - 4n = x_1^2 - 4n$$
$$\Rightarrow x_2 = p - q = \sqrt{x_1^2 - 4n}$$

Conoscendo somma e differenza di p e q si ricava che

$$p = \frac{x_1 + x_2}{2} \qquad q = \frac{x_1 - x_2}{2}$$

Ma questo equivale a fattorizzare.

Ricavare d direttamente da $\langle n, e \rangle$ sembra costoso quanto fattorizzare n. Sostanzialmente, tutta la sicurezza di RSA risiede nel non poter fattorizzare un numero n in tempo polinomiale.

12.2.1 Come fattorizzare un intero

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 sub-esponenziale:

- **General Number Field Sieve** (GNFS) richiede $O\left(2^{\sqrt{b \log b}}\right)$ operazioni per fattorizzare un intero n, con $b = (\log_2 n) + 1$ bit.
- Un attacco **brute force** ne richiede $O(n)$, cioè $O(2^b)$.

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.

La fattorizzazione e il logaritmo discreto non sono problemi NP-hard e si possono risolvere in tempo polinomiale su macchine quantistiche.

12.2.2 Vincoli sui parametri

- Scegliere p e q molto grandi (almeno 1024 bit) per resistere agli attacchi brute force.
- 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 $\frac{p-1}{2}$ e $\frac{q-1}{2}$ siano co-primi.
- Non riusare uno dei primi per altri moduli.
- p e q non devono essere troppo vicini fra loro perché se $n \sim p^2 \sim q^2$ allora $\sqrt{n}$ sarà vicino a p e q: un attacco brute force potrebbe ricercare solo in un intorno di $\sqrt{n}$. Questa idea si può raffinare tenendo conto che

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

Quindi

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

$$\frac{(p+q)^2}{4} - n \text{ è un quadrato perfetto}$$

Si scandiscono gli interi maggiori di $\sqrt{n}$ fino a trovare z t.c. $z^2 - n = w^2$ e si suppone

$$z = \frac{p+q}{2} \qquad w = \frac{p-q}{2}$$

da cui $p = z + w$ e $q = z - w$; la vicinanza tra p e q assicura che non dobbiamo allontanarci troppo da $\sqrt{n}$ per trovare p e q. Per quantificare la distanza fra p e q si dice che è sufficiente che siano distanti più di un fattore logaritmico, cioè che $p - q$ deve crescere più di una funzione poli-logaritmica: $p - q = \Theta(n^\varepsilon)$ con $0 < \varepsilon < 1$.

12.2.3 Attacchi

Attacchi con esponenti bassi

Esponenti e e d bassi sono attraenti perché accelerano cifratura e decifrazione. Ovviamente, d dovrebbe essere scelto sufficientemente grande per evitare attacchi brute force.

Attenzione: se m ed e sono così piccoli che $m^e < n$, allora risulta facile trovare la radice e-esima di c, poiché $c = m^e$ e non interviene la riduzione in modulo!

Attacchi a tempo (DH, RSA)

Si basano sul tempo di esecuzione dell'algoritmo di decifrazione.

L'idea è di determinare d analizzando il tempo impiegato a decifrare: quando si esegue l'algoritmo delle quadrature successive, si esegue una moltiplicazione ad ogni iterazione, più un ulteriore moltiplicazione modulare per ciascun bit uguale a 1 in d. Per rimediare si può aggiungere un ritardo casuale per confondere l'attaccante.

Attacchi legati alla scelta di *e*

Ci sono alcuni valori di e che vanno evitati durante la scelta, per esempio

$$e \neq \frac{\phi(n)+k}{k} \quad \forall\ k \text{ t.c. } k \mid p-1 \text{ e } k \mid q-1 \text{ e t.c. } m \text{ e } n \text{ co-primi}$$

e

$$e \neq q, p$$

Inoltre, e non deve essere piccolo perché può essere usato in un attacco. Supponiamo che ci siano e utenti che hanno scelto lo stesso valore piccolo di e (per esempio, $e = 3$) e anche che questi e utenti ricevano lo stesso messaggio m.

$$c_1 = m^e \ mod\ n_1,\ c_2 = m^e \ mod\ n,\ \ldots,\ c_e = m^e \ mod\ n_e$$

$$\forall i\ 1 \leq i \leq e,\ m < n_i$$

Ipotizziamo che $n_1, n_2, \ldots, n_e$ siano co-primi fra loro. Per il teorema cinese del resto, esiste e si può facilmente calcolare un unico m' tale che

$$m' < n = n_1 \times n_2 \times \cdots \times n_e \qquad m' \equiv m^e mod\ n$$

Inoltre, è noto che

$$m_1 m_2 m_3 \ldots m_e < n_1 n_2 n_3 \ldots n_e = n \Rightarrow m^e < n$$

Applicando la relazione di congruenza si ottiene che $m'\ mod\ n = m^e\ mod\ n$, ma $m' < n$ e $m^e < n$ e quindi la riduzione in modulo non interviene; da ciò si ricava che $m' = m^e$ questo permette di calcolare $m = \sqrt[e]{m'}$.

Usare e piccolo ha comunque i suoi vantaggi, perciò per difendersi da questo attacco si concatena una sequenza di bit casuale (*padding*).

Attacchi con lo stesso valore di n

Supponiamo due chiavi pubbliche $\langle e_1, n\rangle$ e $\langle e_2, n\rangle$ per cui MCD$(e_1, e_2) = 1$. Esisteranno $r, s \in \mathbb{Z}$ t.c. $e_1 r + e_2 s = 1$ che si possono trovare con l'algoritmo di Euclide esteso.

Ipotizziamo che $r < 0$ e $s > 0$ e che il crittoanalista intercetti due crittogrammi $c_1 = m^{e_1} \ mod \ n$ e $c_2 = m^{e_2} \ mod \ n$. A questo punto si imposta

$$m = m^1 = m^{re_1 + se_2} = (m^{e_1} \ mod \ n)^r \cdot (m^{e_2} \ mod \ n)^s =$$

$$= (c_1^r c_2^s) \ mod \ n = \left(\left(c_1^{-1} \right)^{-r} c_2^s \right) \ mod \ n$$

Adesso può ricavare c_1 con l'inverso $c_1^{-1} \ mod \ n$, ma ciò esiste solo se c_1 e n sono co-primi. Dopo si calcola $(c_1^{-1})^{-r}$ e c_2^s con le quadrature successive e infine

$$m = \left(\left(c_1^{-1} \right)^{-r} c_2^s \right) \ mod \ n$$

Tutto ciò si può fare in tempo polinomiale.

Capitolo 13

Protocollo Diffie-Hellman e cifrario di El Gamal

13.1 Protocollo Diffie-Hellman

In un contesto di comunicazione, i cifrari a chiave pubblica vengono sempre utilizzati per lo **scambio delle chiavi**: si parla di *cifrari ibridi*. Se Alice e Bob vogliono utilizzare l'AES per aprire una sessione di comunicazione sicura, devono concordare insieme una chiave simmetrica: useranno l'RSA per scambiarsi la chiave e l'AES per proteggere la loro comunicazione.

Alice:

1. sceglie una chiave k_s per AES
2. la cifra con la chiave pubblica RSA di Bob
3. cifra il messaggio con k_s
4. invia i due crittogrammi a Bob

$$\langle C_{RSA}\left(k_s, k_{\text{Bob}}[pub]\right), C_{AES}(m, k_S)\rangle$$

Bob, invece, decifra il primo crittogramma con $k_{\text{Bob}}[priv]$, trova k_s e decifra il secondo crittogramma.

Poiché la chiave pubblica viene usata per cifrare una sequenza imprevedibile (la chiave deve sempre essere generata casualmente) si è al riparo da attacchi di tipo *choosen plain text*, al quale la crittografia a chiave pubblica è soggetta per sua natura.

Per usare in sicurezza l'RSA è necessario usare chiavi molto lunghe, ciò significa che devono essere eseguite operazioni aritmetiche, elevamenti a potenza, calcolo dell'inverso... di numeri molto grandi; di conseguenza si tratta

di un cifrario molto "costoso". Tuttavia, poiché la chiave dell'AES è una sequenza breve (128 o 256 bit) e deve essere cifrata una volta sola all'inizio della sessione, il costo si ammortizza un po' sulla comunicazione successiva.

Questo modo di usare la crittografia a chiave pubblica (e.g. RSA) per scambiarsi la chiave segreta da usare poi in una crittografia a chiave segreta (e.g. AES) ha una **distribuzione delle responsabilità per la generazione delle chiavi troppo eterogenea**: chi deve generare la chiave privata è sempre il mittente.

In un ambiente distribuito come la rete, si preferisce una divisione più bilanciata delle responsabilità; perciò spesso, a questo approccio ibrido, si preferisce il **protocollo Diffie-Hellman**. Si tratta di un algoritmo per generare e scambiare una chiave di sessione per un cifrario simmetrico. A differenza dell'utilizzo di RSA per lo scambio delle chiavi, il protocollo DH distribuisce le responsabilità più omogeneamente mettendo mittente e destinatario sullo stesso piano per quanto riguarda la generazione della chiave privata.

13.1.1 Funzionamento

L'idea è che Alice e Bob si scambino in chiaro pezzi di informazioni che verranno combinati insieme a informazioni segrete e private da mittente e destinatario per creare la chiave.

1. Alice e Bob si accordano pubblicamente su un numero p molto grande e su un generatore g di $\mathbb{Z}_p^*$ (insieme di tutti i numeri interi minori di p e coprimi con p). Dato che p è primo, esisterà sempre un generatore.

$$\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\} = \{g^k \ mod \ p, \ 1 \leq k \leq p-1\}$$

2. Dopo aver scelto g e p

Alice	Bob
sceglie a caso $1 < x < p-1$	sceglie a caso $1 < y < p-1$
e calcola $A = g^x \ mod \ p$	e calcola $B = g^y \ mod \ p$
manda A a Bob	manda B a Alice
riceve B da Bob e calcola	riceve A da Alice e calcola
$k_s = B^x \ modp = g^{yx} \ mod \ n$	$k_s = A^y \ modp = g^{yx} \ mod \ n$

13.1.2 Attacchi

Attacchi passivi

Nell'attacco passivo, il crittoanalista spia la comunicazione senza intervenire nel processo.

Il crittoanalista conosce p, g, A e B e per calcolare k_s deve trovare x e y. Tuttavia, x e y sono legate alle relazioni $A = g^x \ mod \ p$ e $B = g^y \ mod \ p$ e per trovarle bisognerebbe risolvere un logaritmo discreto, cioè un problema difficile quanto la fattorizzazione. Gli attacchi passivi sono pressoché inutili.

Attacchi attivi

Il protocollo Diffie-Hellman è vulnerabile agli attacchi attivi "*man-in-the-middle*". Quando Alice e Bob si scambiano A e B, Eve li sostituisce sul canale con

$$E = g^z \ mod \ p \qquad z \in (1, p-1)$$

A questo punto entrambi ricevono E (pensando che sia A/B) e si costruiscono rispettivamente

$$k_A = E^x \ mod \ p = g^{xz} \ mod \ p \quad \text{e} \quad k_B = E^y \ mod \ p = g^{yz} \ mod \ p$$

Chiaramente $k_A \neq k_B$ e Eve li conosce entrambi:

$$k_A = A^z \ mod \ p = g^{xz} \ mod \ p \qquad k_B = B^z \ mod \ p = g^{yz} \ mod \ p$$

Perché tutto funzioni correttamente Eve deve intercettare ogni crittogramma diretto in una delle due direzioni, decifrarlo con la chiave corrispondente a quella del mittente, leggerlo, cifrarlo con la chiave corrispondente a quella del destinatario e inviarglielo. Se non facesse così, Alice e Bob non potrebbero decifrare i loro messaggi correttamente e si insospettirebbero.

L'unico modo per proteggersi da questo tipo di attacchi è ricorrere ai **certificati digitali** per autenticare le chiavi pubbliche.

13.2 Cifrario di El Gamal

Si tratta di un cifrario a chiave pubblica alternativo all'RSA e che usa come funzione one-way trap-door il logaritmo discreto.

Bob

- Sceglie p, numero primo molto grande, e g, generatore per $\mathbb{Z}_p^*$

- Sceglie $k_{priv} = \langle x \rangle,\ 2 \leq x \leq p-2$
- Calcola $y = g^x \ mod \ p$
- Pubblica $k_{pub} = \langle p, g, y \rangle$

Alice

- Sceglie un messaggio m tale che $0 \leq m < p$
- Si procura $k_{pub} = \langle p, g, y \rangle$
- Sceglie a caso $2 \leq r \leq p-2$ segreto e calcola $c = g^r \ mod \ p$
- Calcola $d = m \cdot y^r \ mod \ p$
- Invia a Bob la coppia $\langle c, d \rangle$

Bob riceve $\langle c, d \rangle$ e decifra:

$$m = d \cdot (c^x)^{-1} \ mod \ p$$

Correttezza

$$\frac{d}{c^x} \ mod \ p = \frac{y^r \cdot m}{c^x} \ mod \ p = \frac{g^{xr} \cdot m}{(g^r)^x} \ mod \ p = m \ mod \ p = m$$

Sicurezza

Il crittoanalista conosce p, g, y, c e d (tutto tranne r e x). Se conoscesse x, sarebbe in grado di calcolare

$$m = \frac{d}{c^x} \ mod \ p$$

mentre se conoscesse r potrebbe calcolare

$$m = \frac{d}{y^r} \ mod \ p = \frac{y^r \cdot m}{y^r} \ mod \ p = m$$

Il protocollo è sicuro perché r è protetto all'interno di $c = g^r \ mod \ p$ che, essendo una potenza in modulo, necessita del logaritmo discreto (problema non polinomiale) per essere invertito. Stesso discorso vale per x che è protetto dentro la formula $y = g^x \ mod \ p$.

Capitolo 14

Crittografia su curve ellittiche

Oggi si parla di una nuova generazione di cifrari a chiave pubblica facendo riferimento alla crittografia sulle **curve ellittiche**, la quale sta via via sostituendosi ai sistemi basati sull'algebra modulare (RSA, Diffie-Hellman, El Gamal) per i problemi relativi alla pesantezza computazionale delle elaborazioni. Gli attacchi a questi cifrari non sono più attacchi puramente esponenziali, ci sono delle tecniche che permettono di ridurre il costo della fattorizzazione e del calcolo del logaritmo discreto. Quindi si è diffuso un nuovo tipo di crittografia che usa delle operazioni definite sulle curve ellittiche: la funzione one-way trap-door è facile da calcolare, ma per l'inversione si hanno a disposizione solo algoritmi con costo esponenziale nel numero di bit e quindi sono più sicuri.

La crittografia su curve ellittiche (*Elliptic Curve Cryptography*) permette di avere un livello di sicurezza molto più elevato a parità di lunghezza della chiave e di usare chiavi più corte a parità di sicurezza.

14.1 Curve ellittiche

Si chiamano curve ellittiche, ma non sono delle ellissi: sono delle curve algebriche descritte da equazioni che somigliano a quelle usate per calcolare la lunghezza degli archi delle ellissi. Sono uno strumento matematico abbastanza recente; sono studiate dalla metà del XIX secolo, mentre le applicazioni alla crittografia sono più recenti.

A metà degli anni '80, Miller (IBM) e Koblitz (Università di Washington) hanno proposto di prendere i protocolli noti della cifratura a chiave pubblica e modificare gli algoritmi sostituendo le operazioni dell'algebra modulare con le operazioni sui punti delle curve ellittiche.

Definizione 14.1.1. Preso un campo k, una **curva ellittica** è un insieme di punti $(x, y) \in k^2$ tale che

$$y^2 + axy + by = x^3 + cx^2 + dx + e \qquad a, b, c, d, e \in k$$

Se la *caratteristica*[1] di $k \neq 2, 3$ allora è possibile ridurre l'equazione in *forma normale di Weierstrass*:

$$y^2 = x^3 + ax + b \qquad a, b \in k$$

Quindi scriveremo

$$E_k(a, b) = \{(x, y) \in k^2 \mid y^2 = x^3 + ax + b\} \cup \{O\}$$

È interessante lavorare sulle curve ellittiche perché è possibile dare a $E_k(a, b)$ la struttura algebrica di *gruppo abeliano* (non di campo) ed quindi è possibile definirvi un'operazione interna associativa e commutativa con esistenza dell'inverso: combinando due punti della curva tramite questa funzione si ottiene un terzo punto appartenente alla curva. Per poter poter essere un gruppo abeliano è necessario anche l'elemento neutro $\{O\}$, chiamato *punto all'infinito* (a seconda del campo k tale elemento potrebbe essere già presente).

Supponiamo $k = \mathbb{R}$

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

Per poter definire la struttura di gruppo abeliano è richiesto che $x^3 + ax + b$ non abbia radici multiple. Quindi assumiamo che

$$4a^3 + 27b^2 \neq 0$$

in questo caso la cubica non ha radici multiple e perciò è garantita l'esistenza della tangente in ogni punto della curva ellittica.

Nel caso in cui vi siano radici multiple la curva potrebbe assumere una forma a "nodo" o a "cuspide".

Simmetria orizzontale

Assumiamo di avere un punto $P = (x, y) \in E(a, b)$. Allora P soddisfa $y^2 = x^3 + ax + b$ e quindi anche il punto $-P = (x, y) \in E(a, b)$, infatti

$$(-y)^2 = y^2 = x^3 + ax + b$$

$-P$ è detto "*opposto*", l'inverso di P.

Per il *punto all'infinito* si pone $O = -O$.

[1] La **caratteristica** di un campo è il numero di volte che l'elemento neutro moltiplicativo (1) deve essere sommato per ottenere l'elemento neutro additivo (0): nel campo $\mathbb{Z}_p$, la caratteristica è proprio p

14.1.1 Somma sulle curve ellittiche

Si chiama *somma*, ma non ha relazioni con la somma delle coordinate.

Se si prende una curva ellittica e se ne studia l'intersezione si vede che i punti di intersezione sono al massimo tre; infatti, mettendo a sistema una curva ellittica di Weierstrass con una retta, si ottiene un sistema in x^3 che ha al massimo tre soluzioni.

$$\begin{cases} y = mx + q \\ y^2 = x^3 + ax + b \end{cases}$$

Esiste anche il caso in cui c'è una sola soluzione reale e le altre due sono complesse coniugate. Una cosa molto interessante è che se una retta interseca $E(a, b)$ in due punti, allora la interseca anche in un terzo punto: questa caratteristica viene usata per definire l'operazione di "somma".

Definizione 14.1.2. Siano $P, Q, R \in E(a, b)$. Se P, Q, R sono disposti su una retta, si pone $P + Q + R = O$.

Siano $P, Q \in E(a, b)$ e supponiamo che $Q \neq \pm P$. Si consideri la retta $\overline{PQ}$ e il punto generato R con l'intersezione di $E(a, b)$. Si pone $P + Q = -R$ e

$$\begin{cases} R \in E(a, b) \\ -R \in E(a, b) \end{cases}$$

Se $Q = -P$, allora la retta $\overline{PQ}$ è parallela all'asse y e quindi incontra $E(a, b)$ nel punto all'infinito, perciò $P + (-P) = O$ (O è l'elemento neutro).

Nel caso in cui $Q = P$ si hanno due radici uguali nella risoluzione del sistema fra curva e retta. Si considera la tangente alla curva nel punto P (sempre definita per costruzione, in quanto $4a^3 + 27b^2 \neq 0$) e si prende l'opposto del punto di intersezione tra la tangente in P e la curva ellittica (questo punto può essere O).

Proprietà della somma

- **Chiusura**: $\forall\ P, Q \in E(a, b) \Rightarrow P + Q \in E(a, b)$
- **Elemento neutro**: $\forall\ P \in E(a, b) \Rightarrow P + O = O + P = P$
- **Inverso**: $\forall\ P \in E(a, b),\ \exists!\ Q \in E(a, b)$ t.c. $P + Q = O = Q + P$
- **Proprietà commutativa**: $\forall\ P, Q \in E(a, b),\ P + Q = Q + P$
- **Proprietà associativa**: $\forall\ P, Q, R \in E(a, b),\ (P+Q)+R = P+(Q+R)$

Formulazione algebrica

Prendiamo i punti $P = (x_p, y_p)$, $Q = (x_q, y_q)$ e si vuole formulare $S = P + Q$.

Se $Q \neq \pm P \Rightarrow S = (x_s, y_s)$

$$x_s = \lambda^2 - x_p - x_q \quad \text{e} \quad y_s = -y_p + \lambda(x_p - x_s), \qquad \lambda = \frac{y_q - y_p}{x_q - x_p}$$

Se $Q = P$ le formule sono le stesse di prima ma

$$\lambda = \frac{3x_p^2 + a}{2y_p}$$

Si noti che se $y_p = 0$ allora siamo nel caso in cui la retta che passa per i due punti è proprio quella tangente in P che ha come terzo punto il punto all'infinito, perciò $P + P = O$

Se $Q = -P$ allora $P + Q = O$.

14.2 Curve ellittiche su campi finiti

La crittografia ha bisogno di un'aritmetica veloce e assolutamente precisa, senza errori di arrotondamento; di conseguenza, non possono essere usate le curve ellittiche sui reali, ma vengono utilizzate le curve definite su un campo finito:

- **Prime**: $k = \mathbb{Z}_p$, dove p è un numero primo.
- **Binarie**: $k = GF(2^m)$, dove $m \in \mathbb{N}$.

Si osservi che nel caso delle curve binarie, siccome la caratteristica di campo è uguale a due ($k = 2$), la forma normale di Weierstrass non può essere usata.

Per semplicità si considereranno solo le curve prime e poiché la caratteristica di $\mathbb{Z}_p = p$, si pone $p > 3$, primo. E quindi

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

Perché il gruppo di punti della curva sia un gruppo abeliano è necessario che $4a^3 + 27b^2 \neq 0$. A questo punto, la curva non è più una figura continua, ma una nuvola di punti perché i punti stessi sono presi da un campo finito e si lavora solo nel quadrante positivo (intervallo $[0, p-1]$). Inoltre, se $P \in E_p(a, b) \Rightarrow -P = (x, p-y) \in E_p(a, b)$. Si noti che non essendo più un normale piano cartesiano, l'asse di simmetria viene spostato in $y = \frac{p}{2}$.

Formule

Le formule sono identiche a quelle viste sui campi non finiti; le uniche differenze sono la presenza di *mod p* e il fatto che la divisione equivale a moltiplicare per l'inverso a meno della presenza di una semplificazione (e l'inverso si può sempre fare perché si lavora modulo numero primo).

14.2.1 Ordine

La cardinalità del numero di punti appartenenti alla curva è detto **ordine**. Non esiste un modo per definirlo, ma dipende tutto da come è fatta la cubica.

Considerando

$$y^2 \equiv x^3 + ax + b \ mod \ p \quad x \in \mathbb{Z}_p$$

si hanno p valori per x e per ogni x esiste anche il suo opposto, quindi ci si aspetterebbe che il numero di punti $\simeq 2p + 1$. Tuttavia, non tutti i p valori di x della cubica danno origine a un *residuo quadratico*[2]: i valori di x che non danno luogo a un residuo quadratico, non definiscono punti sulla curva.

In un campo finito $\mathbb{Z}_p$, solo $\frac{p-1}{2}$ sono residui quadratici.

Teorema 14.2.1 (Hasse)**.** *Preso N = ordine di una curva prima $E_p(a, b)$*

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

Esempio 14.2.1.

$$y^2 \equiv x^3 + 4x + 4 \ mod \ 5$$

Calcoliamo prima di tutto i punti che possono essere residui quadratici nel campo:

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

Solo 0, 1 e 4 sono residui quadratici.

[2] I **residui quadratici** sono gli elementi del campo $\mathbb{Z}_p$ che ammettono radici nel campo.

$$
\begin{array}{lll}
x = 0,\ y = 4 & \rightarrow & (0,2), (0,3) \in E_5(4,4) \\
x = 1,\ y = 4 & \rightarrow & (1,2), (1,3) \in E_5(4,4) \\
x = 2,\ y = 0 & \rightarrow & (2,0) \in E_5(4,4) \\
x = 3,\ y = 3 & \rightarrow & \text{nessuna soluzione} \\
x = 4,\ y = 4 & \rightarrow & (4,2), (4,3) \in E_5(4,4)
\end{array}
$$

Ordine $= 7 + 1 = 8$ (compreso il punto O).

14.2.2 Funzione one-way trap-door

Esiste una sorta di parallelismo tra le operazioni dell'algebra modulare e le operazioni sulle curve ellittiche:

moltiplicazione	$\rightarrow$	somma di punti
elevazione a potenza	$\rightarrow$	*moltiplicazione scalare* di un punto P della curva per un intero k

e quindi

$$y^k = y \times y \times \cdots \times y \quad \rightarrow \quad kP = P + P + \cdots + P$$

Come l'elevamento a potenza, anche la *moltiplicazione scalare* ha un costo polinomiale. È one-way perché calcolare $Q = kP$ dati k e P è facile e si può fare in $\Theta(\log k)$ operazioni grazie all'algoritmo dei **raddoppi ripetuti**.

Algoritmo generale

$$
\begin{aligned}
k = \sum_{i=0}^{t} k_i 2^i \quad &\rightarrow \quad k = (k_t k_{t-1} \ldots k_1 k_0)_2 \\
&\Rightarrow \#\text{bit} : t + 1 = \lfloor \log_2 k \rfloor + 1
\end{aligned}
$$

1. Si calcolano i punti $2P, 4P, \ldots, 2^t P$ ciascuno come raddoppio del punto precedente. Quindi si eseguono $t = \Theta(\log k)$ raddoppi.

2. Si calcola Q come
$$Q = \sum_{i:k_i=1} 2^i P$$
Si eseguono al più $O(t)$ somme: $O(\log k)$.

Operazione inversa della moltiplicazione

Dati P e Q sulla curva $E_p(a, b)$, trovare se esiste il più piccolo k tale che $Q = kP$.

Sostanzialmente si tratta del problema di trovare $k = log_p Q$ noto come **problema del logaritmo discreto per le curve ellittiche**, nonché un problema difficile in quanto non esistono algoritmi polinomiali e addirittura sub-esponenziali in grado di risolverlo, ne esistono solo esponenziali.

14.3 Protocollo DH su curve ellittiche

Alice e Bob scelgono una curva ellittica che soddisfa la condizione che tutti i suoi punti formino un gruppo abeliano e un punto B della curva di ordine molto grande.

Osservazioni

- Per ordine n di un punto B si intende il più piccolo intero tale che $nB = 0$.
- B "corrisponde" al generatore g in Diffie-Hellman standard.
- Curva e punto B sono pubblici.

Alice	Bob
estrae $n_A < n$ casuale (k_{priv})	estrae $n_B < n$ casuale (k_{priv})
calcola k_{pub}: $P_A = n_A B$	calcola k_{pub}: $P_B = n_B A$
manda P_A in chiaro a Bob	manda P_B in chiaro a Alice
riceve P_B e calcola $S = n_A P_B = n_A n_B B$	riceve P_A e calcola $S = n_B P_A = n_B n_A B$

$$k_{sessione} = x_S \ mod \ 2^{256}$$

Crittoanalista

Il crittoanalista conosce $E_p(a, b), B, P_A, P_B$, ma non può ricavarne niente di interessante in tempo polinomiale: per calcolare S deve trovare n_A tale che $n_A B = P_A$ (oppure $n_B \mid n_B B = P_B$) quindi deve risolvere il problema del logaritmo discreto su curve ellittiche, di cui non ne esistono algoritmi inferiori

al puro esponenziale. Rimane il problema degli attacchi *man-in-the-middle*, perciò è sempre bene usare i certificati.

14.4 Scambio di messaggi cifrati

La prima cosa da fare è trasformare m in un punto P_m della curva ellittica $E_p(a, b)$, il quale verrà trasformato in un altro punto della curva ellittica che sarà il crittogramma.

Vediamo come eseguire $m \to P_m$: considerando l'equazione della curva ellittica

$$y^2 \equiv x^3 + ax + b \ mod \ p$$

potremmo essere tentati di sostituire il messaggio m al posto di x, ma non abbiamo la sicurezza che m sia un residuo quadratico nel campo. In particolare, sappiamo con certezza che $prob(m^3 + am + b$ sia un R.Q.$) \simeq \frac{1}{2}$ e non possiamo accettare un metodo corretto per la metà delle volte (se m non è un residuo quadratico, P_m non è un punto della curva).

Algoritmo di Koblitz

È un algoritmo polinomiale randomizzato che permette di eseguire $m \to P_m$.

Dato $m < p \to P_m \in E_p(a, b)$, si sceglie un intero h tale che $(m+1)h < p$ e successivamente si calcola come ascissa $x = mh + i$ con $0 \leq i < h$. L'idea è di riuscire a fare h tentativi.

```
Koblitz(m, h, a, b, p) //(m + 1)h < p
  for(i = 0; i < h; i++){
    x = mh + i;
    z = (x^3 + ax + b)  mod  p;
    if(z è un residuo quadratico){ //costo polinomiale
      y = √z
      return  P_m = (x, y);
    }
  }
  return failure;
```

L'algoritmo di Koblitz ha una probabilità di fallimento di circa $\left(\frac{1}{2}\right)^h$ e di successo di circa $1 - \left(\frac{1}{2}\right)^h$.

Per risalire a m da x, il destinatario può eseguire l'operazione

$$\left\lfloor \frac{x}{h} \right\rfloor = \left\lfloor \frac{mh + i}{h} \right\rfloor = \left\lfloor m + \frac{i}{h} \right\rfloor = m$$

poiché $0 \leq i < h$.

Scambio di messaggi

Dopo aver trasformato il messaggio in un punto dobbiamo capire come mandarlo. Supponendo di avere una curva $E_p(a,b)$, B di ordine elevato (n) con $B \in E_p(a,b)$, e h per algoritmo di Koblitz, ogni utente genera:

$$k_U[priv] = n_U < n$$

$$k_U[pub] = n_U B$$

Alice mappa m su $P_m \in E_p(a,b)$ (per esempio usando l'algoritmo di Koblitz), dopodiché sceglie un **intero casuale** r e calcola $V = rB$. A questo punto Alice calcola $W = P_m + r \cdot k_{\text{Bob}}[pub]$ (poiché r è un numero casuale, $r \cdot k_{\text{Bob}}[pub]$ è un punto "scelto a caso" su $E_p(a,b)$) e invia a Bob $\langle V, W \rangle$.

Bob riceve $\langle V, W \rangle$ da Alice e decifra P_m:

$$W - k_{\text{Bob}}[priv] \cdot V = (P_m + r \cdot k_{\text{Bob}}[priv]) - k_{\text{Bob}}[priv] \cdot V =$$

$$= P_m + r \cdot n_{\text{Bob}} \cdot B - r \cdot n_{\text{Bob}} \cdot B = P_m$$

Adesso può ricavare m

$$m = \left\lfloor \frac{x}{h} \right\rfloor$$

Crittoanalista

La sicurezza si basa sulla difficoltà del logaritmo discreto su curve ellittiche. Se trova r, decifra:

$$W - k_{\text{Bob}}[pub] = (P_m + r \cdot k_{\text{Bob}}[pub]) - r \cdot k_{\text{Bob}}[pub]$$

Tuttavia, r viaggia ben protetto all'interno di $V = rB$, quindi deve risolvere il logaritmo discreto che ha un costo almeno esponenziale.

Se trova $k_{\text{Bob}}[priv]$ decifra come farebbe Bob. Tuttavia $k_{\text{Bob}}[priv] = n_{\text{Bob}}$ viaggia protetta in $k_{\text{Bob}}[pub] = n_{\text{Bob}} \cdot B$: deve risolvere comunque il logaritmo discreto.

14.5 Motivazioni

Il problema matematico che garantisce la sicurezza della crittografia su curve ellittiche è molto più difficile della fattorizzazione e del calcolo del logaritmo discreto: non esiste alcun limite inferiore esponenziale per i problemi sulle curve ellittiche, però, ad oggi non esistono algoritmi inferiori agli esponenziali.

I **cifrari in algebra modulare** soffrono gli attacchi **index calculus** poiché sfruttano il fatto che il campo $\mathbb{Z}_p$ è sia un gruppo abeliano che un campo (questo permette di dare un concetto di moltiplicazione di punti e facilitare l'index calculus). I cifrari su curve ellittiche non hanno il concetto di moltiplicazione e non è possibile estendervi l'applicazione degli attacchi index calculus.

Algebra modulare (bit del modulo)	Curve ellittiche (bit dell'ordine)
$O\left(2^{\sqrt{b \log b}}\right)$	$O\left(2^{\frac{b}{2}}\right)$

Questi vantaggi si ripercuotono in termini molto pratici sulla dimensione delle chiavi.

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

La tabella riporta la dimensione in bit delle chiavi che garantiscono livelli di sicurezza equivalenti nei tre diversi sistemi, dove due sistemi si considerano di sicurezza equivalente se è richiesto lo stesso costo computazionale per forzarli.

Capitolo 15

Protocolli di identificazione, autenticazione e firma digitale

Identificazione Un sistema di elaborazione, isolato o in rete, deve essere in grado di *accertare l'identità di un utente* che richiede accesso 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.*

Firma digitale

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

Tutto ciò deve essere verificabile da terzi.

Nessuna delle tre funzionalità esclude l'altra, ciascuna estende le precedenti: l'autenticazione del messaggio garantisce l'identificazione del mittente e la firma digitale garantisce l'autenticazione del messaggio. Sono usate per contrastare gli attacchi attivi ed esistono sia sui cifrari simmetrici che asimmetrici.

15.1 Funzioni Hash

Una **funzione hash** $f : X \to Y$ è una funzione tale che

$$n = |X| >> m = |Y|$$

$\exists\ X_1, X_2, \ldots, X_m \subseteq X$ **disgiunti** tale che

$$X = X_1 \cup X_2 \cup \cdots \cup X_m$$

$$\forall\ i,\ \forall\ x \in X_i,\ f(x) = y$$

Sostanzialmente, una funzione hash non è iniettiva, quindi più input corrispondono allo stesso output (più elementi del dominio hanno la stessa immagine).

Una buona funzione hash deve assicurare che **i sottoinsiemi** $X_1, \ldots, X_m$ **abbiano circa la stessa cardinalità**: due elementi estratti a caso da X hanno probabilità $\sim \frac{1}{m}$ di avere la stessa immagine in Y. Un'altra proprietà che viene richiesta è che **elementi di** X **molto simili tra loro appartengano a due sottoinsiemi diversi**: se X è un insieme di interi, due elementi con valori prossimi devono avere immagini diverse. Inoltre deve poter **gestire le collisioni**: l'algoritmo che impiega la funzione hash dovrà affrontare la situazione in cui più di un elemento di X ha lo stesso valore hash Y.

15.1.1 Funzioni hash one-way

Se la funzione è applicata in crittografia, deve soddisfare le seguenti proprietà:

1. $\forall\ x \in X$ è **computazionalmente facile** calcolare

$$y = f(x)$$

2. Proprietà one-way: per la maggior parte degli $y \in Y$ è **computazionalmente difficile** determinare $x \in X$ tale che

$$f(x) = y,\ \text{cioè}\ x = f^{-1}(y)$$

3. Proprietà claw-free: è difficile trovare due elementi $x_1, x_2 \in X$ tali che

$$f(x_1) = f(x_2)$$

15.1.2 Funzioni hash usate in crittografia

MD5 (Message Digest, versione 5)

È una famiglia di algoritmi di cui l'originale non venne mai pubblicato: furono pubblicati MD2 seguito da MD4. Tuttavia, in MD2 e MD4 furono trovate debolezze e, nel 1992, Ron Rivest formulò MD5.

Riceve una sequenza S di 512 bit e produce un'immagine di 128 bit: la sequenza è **digerita** riducendone la lunghezza ad un quarto. In seguito è stato

dimostrato che MD5 non resiste alle collisioni e nel 2004 sono state trovate debolezze severe: oggi la sua sicurezza si considera severamente compromessa. Lo stesso Rivest ha affermato (2005) che era da considerarsi chiaramente forzata dal punto di vista della resistenza alle collisioni.

RIPEMD-160

Versione "matura" delle funzioni MD. Nasce nel 1995 nell'ambito di un progetto dell'Unione Europea. Produce immagini da 160 bit ed è esente dai difetti di MD5.

SHA (Secure Hash Algorithm)

Progettata da NIST e NSA nel 1993, si adotta quando la proprietà di claw-free è cruciale per la sicurezza del sistema.

Opera su sequenze lunghe fino a 2^{64} bit e produce immagini di 160 bit. È una funzione crittograficamente sicura: soddisfa i requisiti delle funzioni hash one-way e genera immagini molto diverse per sequenze molto simili. La prima versione pubblicata (SHA-0) conteneva una debolezza, scoperta in seguito da NSA, che portò alla revisione dello standard.

- 1995: SHA-1, progettato da NSA e raccomandato da NIST.
- 2001: SHA-2, produceva digest più lunghi.
- 2007: a causa degli attacchi su MD5 e SHA-0 e di attacchi teorici su SHA-1, il NIST ha sollecitato proposte per un nuovo algoritmo hash.
- 2012: si è conclusa la valutazione ed è stato selezionata una funzione hash di progettazione non governativa (SHA-3, team di analisti italiani e belgi, rilasciata ufficialmente nel 2015).

SHA-1

Opera su sequenze lunghe fino a $2^{64} - 1$ bit e produce immagini di 160 bit. È molto usata nei protocolli crittografici anche se non è più certificata come standard.

Opera su blocchi di 160 bit, contenuti in un buffer di 5 registri da 32 bit ciascuno, in cui sono caricati inizialmente dei valori pubblici. Il messaggio m viene concatenato con una sequenza di padding che ne rende la lunghezza un multiplo di 512 bit. Il contenuto dei registri varia nel corso dei cicli successivi in cui questi valori si combinano tra loro e con blocchi da 32 bit provenienti da m. Alla fine del procedimento, i registri contengono SHA-1(m).

15.2 Identificazione

15.2.1 Identificazione su canali sicuri

Esempio: accesso di un utente alla propria casella di posta elettronica, o a 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 (admin o hacker).

Il meccanismo di identificazione prevede una cifratura delle password realizzata con funzioni hash one-way.

Ad esempio, in Unix, quando un utente U fornisce per la prima volta la sua password P, il sistema associa a U due sequenze binarie:

- S (seme) prodotta da un generatore pseudocasuale.
- $Q = h(PS)$ con h funzione hash one-way.

Nel file delle password si memorizza $\langle S, Q \rangle$. Quando l'utente si riconnetterà e fornirà nuovamente la password, verrà recuperata S che viene concatenata con la password fornita e viene calcolata $h(PS)$; se risulterà uguale a Q l'identificazione sarà corretta altrimenti no.

Un accesso illecito al file delle password non fornisce informazioni interessanti: è computazionalmente difficile ricavare la password originale dalla sua immagine one-way.

L'aggiunta del seme è molto utile perché, nel caso in cui due utenti usino la stessa password, Q resta comunque diverso.

15.2.2 Identificazione su canali non sicuri

Se il canale è insicuro, la password può essere intercettata durante la sua trasmissione in chiaro. Il sistema non dovrebbe mai maneggiare direttamente la password, ma una sua immagine inattaccabile.

Si adottano strategie che si basano sulla cifratura a chiave pubblica.

Supponiamo che $\langle e, n \rangle$ e $\langle d \rangle$ siano rispettivamente la chiave pubblica e privata dell'utente U che richiede l'accesso ai servizi offerti dal sistema S.

1. S genera un numero casuale $r < n$ e lo invia in chiaro a U.

2. U calcola
$$f = r^d \ mod \ n \quad \text{(firma di } U \text{ su } r\text{)}$$
con la sua chiave privata e lo spedisce a S.

3. S verifica la correttezza del valore ricevuto calcolando e verificando se
$$f^e \ mod \ n = r$$
se cio avviene, l'identificazione ha successo.

Le operazioni di cifratura e decifrazione sono invertite rispetto all'impiego standard nell'RSA: è possibile perché le due operazioni sono commutative

$$(x^e \ mod \ n)^d \ mod \ n = (x^d \ mod \ n)^e \ mod \ n$$

Inoltre, f può essere generata solo da U che possiede $\langle d \rangle$, 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.

Fintanto che il sistema è "onesto" e r è davvero un numero casuale non ci sono problemi; l'utente deve fidarsi del sistema: quest'ultimo potrebbe opportunamente scegliere r in modo da carpire informazioni sulla chiave privata $\langle d \rangle$.

15.3 Autenticazione

Tipicamente per garantire l'autenticazione si usa il MAC (Message Authentication Code): un breve codice associato al messaggio. Per poter calcolare tale codice, il mittente e il destinatario devono concordare una chiave segreta simmetrica k.

Il mittente, quindi, allega al messaggio un MAC $A(m, k)$, allo scopo di garantire la provenienza e l'integrità del messaggio, e spedisce in chiaro la coppia $\langle m, A(m, k) \rangle$, oppure cifra m e spedisce $\langle C(m, k'), A(m, k) \rangle$ (C è una funzione di cifratura, mentre k è la chiave pubblica o segreta del cifrario scelto).

Il destinatario entra in possesso di m (dopo averlo eventualmente decifrato) e, essendo a conoscenza di A e k, calcola $A(m, k)$ confrontando il valore ottenuto con quello inviato dal mittente (per verificare che il MAC ricevuto corrisponda al messaggio a cui risulta allegato). Se la verifica ha successo il messaggio è autenticato, altrimenti scarta il messaggio.

15.3.1 MAC

Il MAC è un'immagine breve del messaggio che può essere 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.

MAC con funzioni hash one-way

$A(m, k) = h(mk)$, con h funzione hash one-way. Risulta computazionalmente difficile per un crittoanalista scoprire la chiave segreta k: h è nota a tutti e m può viaggiare in chiaro o essere scoperto per altra via, ma k viaggia all'interno del MAC e quindi per poterlo recuperare si dovrebbe invertire h.

Il crittoanalista non può sostituire facilmente il messaggio m con un altro messaggio m' perché dovrebbe allegare alla comunicazione di m' il MAC $A(m', k)$ che può essere prodotto solo conoscendo k.

CBC e MAC

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

15.4 Firma digitale

I requisiti fondamentali di una **firma manuale** sono

1. è autentica e non falsificabile, prova che chi l'ha prodotta è chi ha sottoscritto il documento;
2. non è riutilizzabile, è legata strettamente al documento su cui è stata apposta;
3. il documento firmato non è alterabile, chi ha prodotto la firma è sicuro che questa si riferirà solo al documento sottoscritto nella sua forma originale;
4. non può essere ripudiata da chi l'ha apposta, costituisce prova legale di un accordo o dichiarazione.

La **firma digitale** non può consistere semplicemente di una digitalizzazione del documento originale firmato manualmente: un crittoanalista potrebbe

"tagliare" dal documento digitale la parte contenente la firma e "copiarla" su un altro documento. Deve avere una forma che dipenda dal documento su cui viene apposta, per essere inscindibile da questo. Per progettare firme digitali si possono usare cifrari sia i simmetrici che quelli asimmetrici.

15.4.1 Protocollo: messaggio in chiaro e firmato

Si consideri un utente U proprietario della coppia di chiavi $k_U[priv]$ e $k_U[pub]$. C e D sono le funzioni di cifratura e decifrazione di un cifrario asimmetrico.

U genera la firma $f = D(m, k_U[priv])$ per m e spedisce all'utente V la tripla $\langle U, m, f\rangle$. V riceve $\langle U, m, f\rangle$ e verifica l'autenticità della firma f calcolando e controllando che

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

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 C e D in ordine inverso a quello standard, C e D devono essere commutative:

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

Osservazione: sostituire m col suo crittogramma non serve a nulla perché il messaggio in chiaro può essere ottenuto conoscendo la chiave pubblica e la firma, che sono note a tutti.

Il protocollo soddisfa i requisiti della firma manuale:

- f è autentica e non falsificabile: per falsificare la firma occorre conoscere $k_U[priv]$, che è nota solo a U, e inoltre D è one-way.
- Il documento firmato $\langle U, m, f\rangle$ non può essere alterato se non da U, pena la non consistenza tra m e f.
- Poiché solo U può aver prodotto f, U non può ripudiare la firma.
- La firma f non è riutilizzabile su un altro documento $m' \neq m$ poiché è immagine di m.

Altre caratteristiche

- 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.
- È solo uno schema di principio: comporta lo scambio di un messaggio di lunghezza doppia dell'originale poiché la dimensione della firma è paragonabile alla dimensione del messaggio; il messaggio non può essere cifrato perché è ricavabile pubblicamente dalla firma attraverso la verifica di questo.

15.4.2 Protocollo: messaggio cifrato e firmato

Si consideri un utente U proprietario della coppia di chiavi $k_U[priv]$ e $k_U[pub]$. C e D sono le funzioni di cifratura e decifrazione di un cifrario asimmetrico.

U genera la firma $f = D(m, k_U[priv])$ per m e calcola il crittogramma firmato $c = C(f, k_V[pub])$ con la chiave pubblica del destinatario V (si incapsula la firma nel documento cifrato). A questo punto spedisce la coppia $\langle U, c\rangle$ all'utente V. V riceve $\langle U, c\rangle$, decifra il crittogramma $D(c, k_V[priv]) = f$ e cifra tale valore con la chiave pubblica di U ottenendo

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

Se m è significativo attesta l'identità di U: un utente diverso da U ha probabilità praticamente nulla di generare un crittogramma di significato accettabile se "cifrato" con la chiave pubblica di U.

Algoritmo

Immaginiamo che U e V utilizzino l'RSA sia per produrre la firma che per proteggere la loro comunicazione. In questo caso si avranno due coppie di chiavi.

- $\langle d_U\rangle, \langle e_U, n_U\rangle$ chiavi di U.
- $\langle d_V\rangle, \langle e_V, n_V\rangle$ chiavi di V.

U genera la firma del messaggio m

$$f = m^{d_U} \ mod \ n_U$$

e la cifra con la chiave pubblica di V

$$c = f^{e_V} \ mod \ n_V$$

e spedisce a V la coppia $\langle U, c\rangle$.

L'utente V riceve la coppia $\langle U, c\rangle$ e decifra c

$$c^{d_v} \ mod \ n_V = f$$

"decifra" f con la chiave pubblica di U

$$f^{e_U} \ mod \ n_U = m$$

Se m è significativo, conclude che è autentico.

Affinché il procedimento sia corretto è necessario che $n_U \leq n_V$ perché risulti $f < n_V$ e f possa essere cifrata correttamente e spedita a V. Questo impedirebbe a V di inviare messaggi firmati e cifrati a U; per ovviare a questo tipo di problema, ogni utente stabilisce chiavi distinte per la firma e per la cifratura: si fissa pubblicamente H molto grande, ad esempio $H = 2^{1024}$, e si prendono le chiavi di firma $< H$ e le chiavi di cifratura $> H$.

Il valore elevato di H assicura che tutte le chiavi possano essere scelte sufficientemente grandi e, quindi inattaccabili con brute force. Tuttavia, il meccanismo si presta ad altri tipi di attacchi basati sulla possibilità che il crittoanalista si procuri la firma di un utente su messaggi apparentemente privi di senso.

Attacco

Supponiamo che un utente U invii una risposta automatica (ack) al mittente ogni volta che riceve un messaggio m e supponiamo che questo segnale di ack sia il crittogramma della firma di U su m. Un crittoanalista attivo X può decifrare i messaggi inviati a U:

- intercetta il crittogramma c firmato inviato da V ad U, lo rimuove dal canale e lo rispedisce a U, facendogli credere che c sia stato inviato da lui.
- U spedisce automaticamente a X un ack.
- X usa l'ack per risalire al messaggio originale applicando le funzioni del cifrario con le chiavi pubbliche di V e U.

Vediamo il procedimento:

1. V invia il crittogramma c a U

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

$$f = D(m, k_V[priv])$$

2. Il crittoanalista X intercetta c, lo rimuove dal canale e lo rispedisce a U (U crede che c arrivi da X).

3. U decifra c
$$f = D(c, k_U[priv])$$
e verifica la firma con la chiave pubblica di X ottenendo un messaggio
$$m' = C(f, k_X[pub])$$

4. $m' \neq m$ è un messaggio privo di senso, ma il sistema manda l'ack c' a X
$$f' = D(m', k_U[priv])$$
$$c' = C(f', k_X[pub])$$

X usa c' per risalire al messaggio m

1. decifra c' e trova f'
$$D(c', k_X[priv]) = D(C(f', k_X[pub]), k_X[priv]) = f'$$

2. verifica f' e trova m'
$$C(f', k_U[pub]) = C(D(m', k_U[priv]), k_U[pub]) = m'$$

3. da m' ricava f
$$D(m', k_X[priv]) = D(C(f, k_X[pub]), k_X[priv]) = f$$

4. verifica f con la chiave pubblica di V e trova m
$$C(f, k_V[pub]) = C(D(m, k_V[priv]), k_V[pub]) = m$$

Per evitare l'attacco occorre che U blocchi l'ack automatico: l'ack deve essere inviato solo dopo un esame preventivo di m e scartando tutti i messaggi che non si desidera firmare.

15.4.3 Protocollo resistente agli attacchi

Si evita la firma del messaggio: la firma digitale si appone su un'immagine del messaggio ottenuta con una funzione hash one-way e pubblica. La firma non è più soggetta a giochi algebrici.

Il mittente U calcola $h(m)$ e genera la firma

$$f = D(h(m), k_U[priv])$$

Calcola il crittogramma $c = C(m, k_V[pub])$ e spedisce a V la tripla $\langle U, c, f \rangle$.

Il destinatario V attua la fase di decifrazione e verifica: V riceve $\langle U, c, f \rangle$ e decifra c

$$m = D(c, k_V[priv])$$

Calcola separatamente $h(m)$ e $C(f, k_U[pub])$ e se risultano uguali allora il messaggio è autentico.

Il protocollo consente una firma più veloce e lo scambio di una maggiore, ma trascurabile, quantità di dati.

15.4.4 Attacchi man-in-the-middle

Poiché le chiavi di cifratura sono pubbliche e non richiedono un incontro diretto tra gli utenti per il loro scambio, un crittoanalista attivo può intromettersi proprio in questa fase iniziale del protocollo, pregiudicando il suo corretto svolgimento.

Attacco attivo chiamato **man-in-the-middle**: il crittoanalista si intromette nella comunicazione tra U e V e si comporta agli occhi di U come se fosse V (e viceversa), intercettando e bloccando le comunicazioni di U e V.

Il crittoanalista X devia le comunicazioni che provengono da U e V e le dirige verso se stesso:

- U richiede a V la sua chiave pubblica.
- X intercetta la risposta contenente $k_V[pub]$ e la sostituisce con la sua $k_X[pub]$.
- X si pone in ascolto in attesa dei crittogrammi da U a V cifrati con la sua chiave pubblica $k_X[pub]$.
- X rimuove tutti questi crittogrammi, li decifra con $k_X[priv]$, se li legge per bene, li cifra con $k_V[pub]$ e li spedisce a V.

- U e V non si accorgono della presenza di X se egli è sufficientemente veloce nel rimuovere, decifrare e reinserire sul canale i crittogrammi.

15.4.5 Certification Authority

Per evitare attacchi *man-in-the-middle* si ricorre alle cosiddette **certification authority**: sono delle infrastrutture, degli 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 certification authority autentica l'associazione ⟨`utente,chiave pubblica`⟩ 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 mantiene un archivio di chiavi pubbliche sicuro, accessibile a tutti e protetto da attacchi di scrittura non autorizzati.

La chiave pubblica della CA è nota agli utenti che la mantengono protetta da attacchi esterni e la utilizzano per verificarne la firma sui certificati.

Certificato digitale

Un certificato digitale contiene:

- un'indicazione del suo formato (numero di versione);
- il nome della CA che lo ha rilasciato;
- un numero seriale che lo individua univocamente all'interno della CA;
- la specifica dell'algoritmo usato dalla CA per creare la firma elettronica;
- il periodo di validità del certificato (data di inizio e data di fine);
- 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 U vuole comunicare con V può richiedere $k_V[pub]$ alla CA che risponde inviando a U il certificato digitale di V e, poiché U conosce $k_{CA}[pub]$, può controllare l'autenticità del certificato verificandone il periodo di validità e la firma prodotta dalla CA su di esso. Se tutti i controlli vanno a buon fine, $k_V[pub]$ nel certificato è corretta e U avvia la comunicazione con V. Il crittoanalista potrebbe falsificare la certificazione ma si assume che la CA abbia un archivio di chiavi inattaccabile.

15.4.6 Protocollo: messaggio cifrato, firmato in hash e certificato

Il mittente U si procura $cert_V$ di V, calcola $h(m)$ e genera la firma

$$f = D(h(m), k_U[priv])$$

Calcola il crittogramma $c = C(m, k_V[pub])$ e spedisce a V la tripla

$$\langle cert_U, c, f \rangle$$

$cert_U$ contiene la chiave $k_U[pub]$ e la specificazione della funzione h utilizzata.

Il destinatario V riceve $\langle cert_U, c, f \rangle$ e verifica l'autenticità di $cert_U$ (e dunque di $k_U[pub]$) utilizzando la propria copia di $k_{CA}[pub]$. V decifra il crittogramma c mediante la sua chiave privata e ottiene

$$m = D(c, k_V[priv])$$

V verifica l'autenticità della firma apposta da U su m controllando che risulti

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

Conclusioni

Il punto debole di questo meccanismo è rappresentato dai certificati revocati, cioè non più validi: le 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 cruciali per la sicurezza.

Attacchi *man-in-the-middle* possono essere evitati se si stabilisce un incontro diretto tra utente e CA nel momento in cui si istanza il sistema asimmetrico dell'utente.

Capitolo 16

Sistemi Zero-K e SSL

16.1 Sistemi "Zero-Knowledge"

L'obiettivo è permettere ad un determinato Prover di convincere un Verifier, tramite una serie di sfide, di possedere una caratteristica/conoscenza/capacità mostrando al Verifier la stessa e sola caratteristica in sé.

Esempio

Supponiamo che Peggy sostenga di contare i granelli di sabbia di una qualunque spiaggia soltanto osservandoli e Victor vuole controllare se quanto sostiene è vero: vuole verificare l'affermazione di Peggy lasciandole una probabilità arbitrariamente piccola di convincerlo di qualcosa che non è vero.

Per semplicità si supponga che Victor si accontenti che Peggy dica solo se il numero di granelli nella spiaggia è pari o dispari.

Nella fase di preparazione Peggy comunica b_0: parità dei granelli di sabbia a Victor.

```
Protocollo
    for (i = 1 to k){ //k scelto da V
        P si volta
        V sceglie e in {0,1} a caso
        if (e == 0) V toglie un granello di sabbia
        V chiede a P il nuovo valore b[i]
        if ((e == 0 && b[i] != b[i-1]) ||
            (e == 1 && b[i] == b[i-1]))
                continua alla prossima iterazione
        else return "P è un impostore"
    }
```

Supponendo che il Prover sia un impostore, la probabilità di dare risposte giuste è equivalente a rispondere correttamente 0 o 1, quindi $\frac{1}{2}$, per tutte le k volte. Perciò, la probabilità che il Prover dia sempre risposte corrette e riesca ad ingannare il Verifier è $\left(\frac{1}{2}\right)^k$ e di conseguenza la probabilità che P dica il vero è $1 - \left(\frac{1}{2}\right)^k$.

Si noti che k viene scelto da Victor, è lui a decidere quante sfide deve superare Peggy per convincerlo. Quindi, la probabilità di inganno è funzione di un parametro k scelto dal Verifier.

Principi generali dei protocolli Zero-K

- **Completezza**: se P è onesto e la sua affermazione è vera, V deve accettare sempre la dimostrazione.
- **Correttezza**: se P è disonesto e la sua affermazione è falsa, V può essere ingannato con probabilità $\leq \left(\frac{1}{2}\right)^k$ con k scelto da V.
- **Conoscenza zero**: se l'affermazione di P è vera, V (anche se disonesto) non può acquisire alcuna informazione se non la veridicità di quanto affermato da P.

La finalità di questi protocolli è far si che P possa dimostrare a V che è il proprietario di una certa chiave privata senza effettuare alcuna operazione su dati consegnati ad esso da parte di V.

16.1.1 Protocollo Fiat-Shamir

È un protocollo di identificazione Zero-K con l'obiettivo che P dimostri a V la sua identità senza svelare nessun'altra informazione. Si basa sulla difficoltà di calcolo della $\sqrt{}$ in modulo n, con n composto.

$$t \equiv s^2 \ mod \ n \quad n \text{ composto}$$

V conosce t e n, mentre P convince V di conoscere $\sqrt{t}$.

Preparazione

P sceglie p e q primi molto grandi, calcola $n = p \cdot q$, sceglie $s < n$ e calcola $t = s^2 \ mod \ n$. P rende infine nota la coppia $\langle t, n \rangle$ (chiave pubblica) e mantiene segreto $\langle p, q, s \rangle$ (chiave privata).

Conoscendo solo $\langle t, n \rangle$ non è possibile calcolare in tempo polinomiale nessuno dei fattori contenuti in $\langle p, q, s \rangle$.

Protocollo

Ripetizione per k volte di

1. V chiede a P di iniziare un'iterazione.
2. P genera un intero $r < n$ casuale $u = r^2 \ mod \ n$ e lo comunica a V.
3. V genera un $e \in \{0, 1\}$ casuale e lo comunica a P (è un passo cruciale perché se P conosce come V genera e, può sfruttarlo nelle sue risposte).
4. P calcola $z = r \cdot s^e \ mod \ n$ e comunica z a V. Si noti che

$$z = \begin{cases} r \ mod \ n & \text{se } e = 0 \\ r \cdot s \ mod \ n & \text{se } e = 1 \end{cases}$$

   Si noti anche che in questo protocollo, r è generato dallo stesso utente che vi applica la chiave segreta (P), perciò, V non può ingannare P mandandogli un r studiato appositamente per ricavare la chiave dopo l'operazione $r \cdot s \ mod \ n$. Difatti, r è totalmente casuale e applicargli s significa applicare una maschera ad un numero casuale che nessuno è in grado di togliere se non conosce tale maschera.
5. V calcola $x = z^2 \ mod \ n$ e controlla se $x = ut^e \ mod \ n$ infatti

   $z^2 \ mod \ n = (rs^e)^2 \ mod \ n = r^2(s^e)^2 \ mod \ n = u(s^2)^e \ mod \ n = ut^e \ mod \ n$

   Se la condizione è verificata riparti dal passo 1, altrimenti concludi con P è un imbroglione.

Completezza

$$x = \begin{cases} ut^e \ mod \ n = u \ mod \ n & \text{se } e = 0 \\ ut^e \ mod \ n = (rs^e)^2 \ mod \ n & \text{se } e = 1 \end{cases}$$

Se P conosce r e s passa tutte le sfide e V accetta la dimostrazione.

Correttezza

Supponiamo che V mandi sempre $e = 1$. P si aspetta di ricevere sempre $e = 1$, al passo 2 sceglie r a caso e invia a V

$$u = \frac{r}{t} \ mod \ n = r^2 \cdot t^{-1} \ mod \ n$$

Al passo 4, P invia a V $z = r \ mod \ n$.

A questo punto V va a verificare se $x = z^2 \ mod \ n$, ovvero se $x = ut^e = ut$. Sappiamo che $x = z^2 \ mod \ n = r^2 \ mod \ n$, quindi si ricava che

$$ut = \left(\frac{r^2}{t}\right) \cdot t \ mod \ n = r^2 \ mod \ n$$

Di conseguenza si ottiene che $x = ut$.
Osservazioni

- P disonesto riesce a ingannare V se prevede correttamente il bit e inviato da V a ogni iterazione.
- Poiché e è generato casualmente, le previsioni di P sono corrette con probabilità $\frac{1}{2}$ ad ogni iterazione; quindi la probabilità di iganno è $\left(\frac{1}{2}\right)^k$.

Conoscenza-zero

Si tratta di una dimostrazione molto complessa che fa uso di un simulatore probabilistico (Macchina di Turing probabilistica).

16.2 Protocollo SSL

Il protocollo SSL (**Secure Socket Layer**) è alla base dei protocolli più diffusi nelle comunicazioni sicure. Garantisce **confidenzialità** e **affidabilità** alle comunicazioni su Internet, proteggendole da intrusioni, modifiche o falsificazioni. È stato sviluppato da Netscape per effettuare comunicazioni sicure con il protocollo HTTP e la prima versione è stata rilasciata nel 1994: l'idea era di permettere la comunicazione tra computer che non conoscono le reciproche funzionalità. Oggi è stato sostituito dal TLS (**Transport Layer Security**), una variante di SSL con piccole modifiche.

Supponiamo che un un'utente U desideri accedere via Internet a un servizio offerto dal sistema S. La **confidenzialità** è garantita dal fatto che la trasmissione è cifrata mediante un *sistema ibrido*: cifrario asimmetrico per costruire e scambiare le chiavi segrete di sessione e cifrario simmetrico per criptare dati nelle comunicazioni successive. L'**autenticazione** dei messaggi accerta l'identità dei due partner attraverso un cifrario asimmetrico o facendo uso di certificati digitali e inserendo nei messaggi stessi un apposito MAC (che usa una funzione hash one-way crittograficamente sicura).

SSL si innesta tra un protocollo di trasporto e un protocollo di applicazione, ed è completamente indipendentemente da quest'ultimo. L'HTTPS è la combinazione tra SSL e HTTP.

Si organizza su due livelli: **SSL record** e **SSL handshake**. 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 responsabile dell'instaurazione e del mantenimento dei parametri usati da SSL record. Permette all'utente di autenticarsi, negoziare gli algoritmi di cifratura e firma e stabilire le chiavi per i singoli algoritmi crittografici e per il MAC.

Grazie all'SSL handshake (meccanismo crittografico) si crea un canale *sicuro*, *affidabile* e *autenticato* tra utente e sistema, entro il quale SSL Record fa viaggiare i messaggi. Tramite SSL Record si realizza *fisicamente* un canale (meccanismo di rete): utilizza la cipher suite stabilita da SSL Handshake per cifrare e autenticare i blocchi di dati, prima di spedirli attraverso il protocollo di trasporto sottostante.

16.2.1 SSL Handshake

Consiste in una serie di scambi di messaggi preliminari (*handshake*), indispensabili per la creazione del canale sicuro, tramite i quali 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.

1. U manda a S `client hello` con cui richiede la creazione di una connessione SSL, specificando le prestazioni di sicurezza che desidera siano usate durante la connessione (cifrari e meccanismi di autenticazione che U può supportare) e invia una serie di byte casuali.

2. Il sistema S riceve il messaggio di `client hello` e risponde con un `server hello` che specifica una *cypher suite* che anch'esso è in grado di supportare e che contiene una nuova sequenza di byte casuali. Se U non riceve il messaggio di `server hello` interrompe la comunicazione.

3. S si autentica con U inviandogli il proprio certificato digitale e gli eventuali altri certificati della catena di certificazione dalla sua CA fino alla CA radice. Se i servizi offerti da S devono essere protetti negli accessi, S può richiedere a U di autenticarsi inviando il suo certificato digitale; avviene raramente: poiché la maggior parte degli utenti non ha un certificato personale, di solito il sistema si accerta dell'identità dell'utente in un secondo momento.

4. S invia il messaggio `hello done` con cui sancisce la fine degli accordi sulla *cypher suite* e sui parametri crittografici a essa associati.

5. Per accertare l'autenticità del certificato ricevuto da S, l'utente U 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 apposta dalla CA sia autentica.

6. L'utente U costruisce un *pre-master secret* costituito da una nuova sequenza di byte casuali (genera un numero segreto), lo cifra con il cifrario a chiave pubblica su cui si è accordato con S e lo spedisce a S. Dopodiché combina il *pre-master secret* con alcune stringhe note e con i byte casuali presenti sia nel `client hello` che in quello di `server hello`, vi applica delle funzioni hash one-way secondo una combinazione opportuna e ottiene il *master secret*.

7. S decifra il crittogramma contenente il *pre-master secret* ricevuto da U e calcola il *master secret* mediante le stesse operazioni eseguite da U al passo 6 (dispone delle stesse informazioni).

8. 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 il *master secret* e tutti i messaggi scambiati fino a quel momento (SSL-*history*). S controlla il certificato di U e verifica autenticità e correttezza della SSL-*history*; in presenza di anomalie, la comunicazione viene interrotta.

9. Le due parti si scambiano il primo messaggio protetto (*finished*) tramite *master secret* e la *cipher suite* su cui si sono accordati su cui si sono accordati. Il messaggio viene prima costruito da U e spedito a S e poi viceversa: nei due invii la struttura del messaggio è la stessa, ma cambiano le informazioni in esso contenute. La costruzione del messaggio avviene in due passi

   (a) all'inizio si concatenano il *master secret*, tutti i messaggi di *handshake* scambiati fino a quel momento e l'identità del mittente (U o S);

   (b) la stringa ottenuta viene trasformata applicando le funzioni SHA-1 e MD5: si ottiene una coppia di valori che costituisce il messaggio *finished*.

   Il messaggio è diverso nelle due comunicazioni perché S aggiunge ai messaggi di *handshake* anche il messaggio *finished* ricevuto da U. Il destinatario della coppia non può invertire la computazione precedente in

quanto generata da funzioni one-way, perciò deve ricostruire l'ingresso delle due funzioni SHA-1 e MD5, le ricalcola e controlla che la coppia generata coincida con quella ricevuta.

A questo punto sia C che S usano il master secret per costruire una propria tripla contenente: la chiave segreta da adottare nel cifrario simmetrico, la chiave per l'autenticazione del messaggio (costruzione del MAC) e la sequenza di inizializzazione per cifrare in modo aperiodico messaggi molto lunghi.

Le triple di U e 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 da *SSL record* tramite una **frammentazione** in blocchi. Ciascun blocco viene numerato, compresso e autenticato con l'aggiunta di un MAC, cifrato mediante il cifrario simmetrico concordato nell'handshake e trasmesso utilizzando il protocollo di trasporto sottostante. Il destinatario decifra e verifica l'integrità dei messaggi attraverso il MAC, decomprime e riassembla i blocchi in chiaro e li consegna all'applicazione.

16.2.2 Sicurezza

Nei passi di *hello* i due partner creano e si inviano due sequenze casuali per la costruzione del *master secret*, che risulta così diverso in ogni sessione di SSL. Un crittoanalista non può 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 concatenando il contenuto del blocco, il numero del blocco della sequenza, la chiave del MAC e alcune stringhe note e fissate a priori.

Dato che i MAC sono cifrati insieme al messaggio, un crittoanalista non può alterarli senza avere forzato prima la chiave simmetrica e di cifratura; un attacco volto a modificare la comunicazione tra i 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 *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 costruire il *master secret*, su cui si fonda la costruzione di tutte le chiavi segrete adottate nelle comunicazioni successive: quindi, solo il sistema S, quello 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 effettivamente spediti da esso. Se ciò non si verifica, S deduce che il protocollo è stato alterato (casualmente o maliziosamente con un attacco *man-in-the-middle*) e interrompe la comunicazione.

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*, # carta di credito).

Generazione delle sequenze casuali

Le tre sequenze generate da U e S e comunicate nei messaggi di `client hello`, `server hello`, *pre-master secret* sono usate per creare il *master secret*, quindi per generare le chiavi segrete di sessione. La sequenza corrispondente al *pre-master secret* viene generata da U e comunicata per via cifrata 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* e ha lo scopo di consentire un ulteriore controllo sulle comunicazioni precedenti per garantire che queste siano avvenute correttamente, che U e S dispongano dello stesso *master secret* e che la comunicazione non sia stata oggetto di un attacco attivo.

Capitolo 17

Quantum Distribution Key

Il **Quantum Distribution Key** (**QDK**) è un protocollo per la distribuzione quantistica delle chiavi e si pone come alternativa al protocollo Diffie-Hellman. I protocolli per la distribuzione quantistica delle chiavi sono già in uso, non hanno bisogno del computer quantistico per funzionare; si usano insieme al One-Time Pad e permettono di creare e scambiare lunghissime chiavi binarie in massima sicurezza.

La meccanica quantistica non può essere utilizzata per lo scambio dei messaggi perché utilizza dei principi che permettono di verificare se c'è stata un'intrusione sul canale: una volta che Alice e Bob hanno costruito una chiave, prima di usarla eseguono una verifica per controllare se durante lo scambio di chiavi qualcuno ha provato ad intercettarle; in caso affermativo la chiave viene scartata prima che venga usata. Siccome la verifica dell'intrusione avviene a comunicazione conclusa, è chiaro che non sarebbe efficace usarlo per lo scambio di messaggi.

Inoltre, esiste la crittografia *post quantistica* che riguarda la cifratura a chiave pubblica e l'obiettivo di questo tipo di crittografia è andare ad individuare dei cifrari a chiave pubblica che rimangono difficili anche per le macchine quantistiche.

Principi della meccanica quantistica

- **Sovrapposizione di stati**: proprietà di un sistema quantistico di trovarsi in più stati contemporaneamente.
- **Decoerenza**: l'osservazione o la misurazione di un sistema quantistico disturba il sistema stesso che va a perdere la proprietà della sovrapposizione di stati collassando in uno stato singolo.

- **No-cloning**: impossibilità di duplicare un sistema quantistico conservando nella copia lo stato quantico dell'originale; sostanzialmente è impossibile copiare uno stato quantistico non noto.
- **Entanglement**: possibilità che due o più elementi si trovino in stato quantico correlato fra loro in modo che, pur se portati a grande distanza, mantengano la loro correlazione.

17.1 Protocollo BB84

È un protocollo per scambiare le chiavi mediante invio di **fotoni polarizzati**. I fotoni sono privi di massa e si propagano alla velocità della luce. Un fotone può avere diversi stati di polarizzazione (o piani di oscillazione del campo elettrico), a noi ne interessano quattro: verticale, orizzontale, a $+45°$ e a $-45°$. Questi stati vengono divisi in due coppie, la **base di polarizzazione ortogonale** e la **basi di polarizzazione diagonale**. In ciascuno di questi stati viene codificato un bit di informazione, per esempio:

- polarizzazione verticale $\rightarrow 0$
- polarizzazione orizzontale $\rightarrow 1$
- polarizzazione a $+45°$ $\rightarrow 0$
- polarizzazione a $-45°$ $\rightarrow 1$

Perché usare due basi e non una sola? Scegliendo una base sola, non saremmo in grado di misurare con precisione lo stato di polarizzazione del fotone. Se Bob conosce la base di preparazione, la misura del fotone sarà accurata e conoscerà con certezza il bit di informazione; se invece la base è sconosciuta, è possibile distinguere solo due stati di polarizzazione nell'ambito della stessa base: se Bob sceglie una base diversa da quella usata da Alice, il risultato della misura è probabilistico.

Osservazione: il protocollo BB84 non dipende in alcun modo da difficoltà di particolari problemi matematici e nemmeno dal fatto che un giorno potrebbe essere dimostrato che $P = NP$.

17.1.1 Apparecchiatura

Il protocollo richiede due canali: la prima comunicazione avviene sul canale quantistico, dove viaggiano i fotoni che incapsulano il bit casuale scelto da

Alice. La seconda fase di comunicazione avviene sul canale standard e serve a comunicare la base utilizzata per misurare e per inviare.

L'**OPG** è un'apparecchiatura che permette di emettere un fotone alla volta. Un'altra componente, la **PC**, consente di imporre al fotone una certa polarizzazione. Alice utilizzerà un *random number generator* per scegliere (casualmente) se mandare 0 o 1 e in quale base mandarlo.

Bob sceglie casualmente una base tramite un *random number generator* e usa un **deviatore** per scegliere il **PBS** (*beam splitter polarizzante*) corretto: serve a misurare il fotone per determinarne la polarizzazione; si noti che risponde correttamente soltanto se la base alla quale fa riferimento è la stessa che Alice aveva usato al momento dell'invio.

PBS

Sia S l'asse di polarizzazione del PBS e F la direzione di polarizzazione del fotone. Dopo che il fotone è entrato nel beam splitter viene deviato in una delle due uscite possibili: A (*assorbimento*) e R (*riflessione*).

- Il fotone viene inviato all'uscita A con probabilità $\cos^2\Theta$ e assume polarizzazione S.
- Il fotone viene inviato all'uscita R con probabilità $\sin^2\Theta$ e assume polarizzazione $\perp$ a S.

Θ è l'angolo tra le due polarizzazioni.

- $\Theta = 0°$ ($F \equiv S$): il fotone esce da A, con polarizzazione $S = F$.
- $\Theta = 90°$ ($F \perp S$): il fotone esce da R, con polarizzazione $S^\perp = F$.
- $\Theta = \pm 45° \Rightarrow \cos^2\Theta = \sin^2\Theta = \frac{1}{2}$: il fotone esce con pari probabilità da A o R e la sua polarizzazione cambia $(S, S^\perp)$.

Fenomeno quantistico: la lettura attraverso il PBS *distrugge* lo stato quantistico precedente.

Tutti quei bit ottenuti tramite una base diversa da quella usata per generarlo, essendo il risultato completamente casuale, verranno scartati e non saranno inclusi nella chiave. Poiché la base scelta per interpretare il fotone è casuale, si ha il 50% di probabilità di trovare la base corretta, quindi solo metà dei bit inviati sotto forma di fotoni da Alice verranno effettivamente "percepiti" come parte della chiave da Bob.

Alla fine di questo procedimento, Alice e Bob si salvano quali basi sono state scelte per polarizzare e quali per interpretare e se le comunicheranno sul canale standard.

Fasi del BB84

1. Alice invia una sequenza S_A di bit codificati nelle polarizzazioni dei fotoni sul canale quantistico.

2. Bob interpreta S_A con le sue basi che ha scelto casualmente e ottiene S_B (dove le basi coincidono, le misure sono uguali).

3. Sul canale standard, Bob comunica ad Alice le basi scelte e Alice indica quali sono comuni alle sue.

4. Alice e Bob sacrificano una porzione S''_A, S''_B di S'_A e S'_B in posizioni prestabilite, comunicandole sul canale standard (si passano alcuni bit di S'_A e S'_B). Se $S''_A \neq S''_B$ interrompono la comunicazione, altrimenti usano $S'_A \backslash S''_A = S'_A \backslash S''_A$ come chiave.

In assenza di interferenze di Eve, Alice e Bob possiedono una sottosequenza identica ($S'_A = S'_B$), formata dai bit codificati e decodificati con basi comuni.

$$|S'_A| = |S'_B| \simeq \frac{|S_A|}{2}$$

Quantum Bit Error Rate

Il **Quantum Bit Error Rate** (QBER) è la percentuale di bit errati (dovuti ad errori dell'apparato sperimentale e non da intrusioni). Nel confronto tra le due sequenze che Alice e Bob sacrificano per verificare se c'è stato un crittoanalista, se la percentuale di errori è maggiore di QBER allora concludono che Eve è stata sul canale, altrimenti l'errore viene attribuito all'apparato sperimentale: usano un codice a correzione di errori per ricostruire una chiave corretta.

Nota bene: Eve può comunque non far notare il suo passaggio sul canale quantistico intercettando solo pochi fotoni così da far attribuire la colpa degli errori all'apparato.

Per evitare che Eve entri in possesso di qualche bit della chiave senza che Alice e Bob lo sappiano, le sequenze ottenute vengono inserite in funzioni hash e come chiavi vengono usate le immagini hash.

Protocollo

Canale quantistico

$S_A[1, n] \rightarrow$ sequenza iniziale di bit da cui verrà estratta la chiave (rappresentata con un codice a correzione di errori).

```
for i=1 to n
    Alice: sceglie una base a caso, codifica S_A[i],
           invia il fotone a Bob

    Eve intercetta il fotone, lo misura con una sua base,
           lo invia a Bob e calcola S_E[i]

    Bob: sceglie una base a caso, interpreta il fotone
           ricevuto e costruisce S_B[i]
```

Canale standard

QBER è la % di errore dovuti al rumore, mentre h è una funzione hash crittografica.

Bob comunica ad Alice la sequenza di basi scelte.
Alice comunica a Bob le basi comuni.

1. estraggono S'_A e S'_B corrispondenti alle basi comuni e da queste estraggono due sottosequenze in posizioni concordate $\rightarrow S''_A, S''_B$.
2. Si scambiano S''_A e S''_B. Se la % di bit $\neq$ è $>$ QBER $\Rightarrow$ Stop.
3. Altrimenti calcolano $S'_A \backslash S''_A$ e $S'_B \backslash S''_B$, le decodificano con il codice a correzione di errori $\Rightarrow$ ottengono una sequenza comune S_C.

   Attenzione: Eve potrebbe conoscere alcuni bit di S_C.

4. Alice e Bob calcolano $k = h(S_C)$ e usano k come chiave.

Capitolo 18

Bitcoin e Cryptovalute

Nasce nel 2008 dopo l'uscita dell'articolo *"Bitcoin": a peer to peer Electronic Cash System* di Satoshi Nakamoto (pseudonimo). Nel 2009 viene rilasciato il primo software che permette di utilizzare il suddetto protocollo di pagamento e il 3 Gennaio dello stesso anno nasce il "*blocco genesi*" della **block chain** che porta alla nascita dei primi 50 BTC (ogni volta che viene aggiunto un blocco alla block chain viene creata nuova moneta).

La prima transazione eseguita è del 12 Gennaio 2009: Satoshi manda 10 Bitcoin a Hal Finney; mentre la prima transazione commerciale è avvenuta nel 2010.

Idea

Ogni utente A possiede una coppia di chiavi $\langle k_A[pub], k_A[priv]\rangle$ appartenenti a un sistema crittografico basato sulle curve ellittiche e per ogni transazione viene generata una nuova coppia di chiavi. A partire dalla chiave pubblica si calcola l'**indirizzo**, ossia l'identificatore di A, che serve per inviare pagamenti all'utente e verificarne la firma. La chiave privata serve invece per firmare le transazioni. Il **wallet** è l'insieme delle credenziali che attestano la proprietà in BTC di un utente ($\langle$indirizzo, chiave privata$\rangle$ e software di gestione).

18.1 Transazione

Una transazione è un movimento di Bitcoin da un utente ad un altro. Supponendo che Alice voglia mandare x BTC a Bob, la transazione è un messaggio m

$$m = \text{add}_A - x - \text{add}_B$$

firmato tramite $h = \text{SHA-256}(m)$

$$f = D(h, k_A[priv])$$

Dopodiché A diffonde in rete (**broadcast**) la coppia $\langle m, f \rangle$. Tuttavia, essendo in un sistema distribuito senza un'unità centrale di controllo, Bob deve aspettare ($\sim$ 10 minuti) che la rete convalidi la transazione.

Inoltre, non è nemmeno sicuro che la transazione sia parte della block chain giusta dopo 10 minuti; esiste il fenomeno dei **blocchi orfani**: due utenti potrebbero aggiungere un blocco alla block chain contemporaneamente e a questo punto solo il blocco che sviluppa una catena più consistente viene considerata "la vera" block chain. Tutti i blocchi che facevano parte del blocco orfano sono da considerarsi annullati. Tipicamente, dopo l'aggiunta di sei blocchi dopo quello della transazione in questione, essa può essere considerata valida (un'ora di tempo).

Struttura di una transazione

Una transazione si usa come banconota: se qualcuno riceve un input di 0.8 BTC è come se ricevesse una banconota da 0.8 BTC; quindi se deve pagare a qualcuno 1.1 BTC deve possedere una "banconota" da 0.8 BTC e una da 0.3 BTC. Alternativamente può usare una banconota da 2 BTC mandando 1.1 BTC al destinatario e 0.9 BTC a un wallet secondario che usa come "banca resti".

Tipicamente nelle transazioni deve valere che $\sum input \geq \sum output$ e nel caso in cui sia strettamente maggiore, la differenza andrà a colui che confermerà il blocco della block chain che la contiene: l'utente dovrebbe lasciare una piccola differenza per incentivare i miner a confermare proprio il suo blocco.

Validazione di una transazione

I nodi controllano un vario numero di transazioni e controllano che siano valide (controllano in particolare se esiste la transazione di input per verificare se è disponibile la liquidità in BTC). Quando si sono raccolte un discreto numero di transazioni valide (centinaia o migliaia), esse diventano un blocco di transazioni e gli utenti cercano di aggiungerlo alla block chain.

Aggiungere il blocco alla block chain è un processo molto laborioso poiché i vari nodi della catena si mettono d'accordo sul fatto che le transazioni siano valide o meno e questo richiede un po' di tempo; questo fa nascere il problema del double spending: un utente furbetto potrebbe approfittarsi di questi 10 minuti di accordi per effettuare due transazioni e "pagarne solo

uno". Supponiamo che egli abbia 2 BTC ed esegua due pagamenti da 2 BTC uno subito dopo l'altro. Al momento in cui devono essere validate le suddette transazioni in un blocco, nessuno ha rimosso i 2 BTC dal suo wallet perché il blocco dove esse sono contenute non è stato ancora confermato e quindi entrambe le richieste di pagamento risultano valide sulla carta. Tuttavia, l'architettura della block chain mitiga bene questo problema.

18.2 Architettura della block chain

Nel blocco i-esimo della block chain sono contenuti vari componenti:

- Le transazioni da confermare (a cui viene aggiunta la transazione "premio" destinata a chi conferma il blocco).
- La funzione hash del blocco $i-1$. È importante perché l'i-esimo blocco deve essere collegato al blocco $(i-1)$-esimo.
- Il numero intero **nonce** che è sconosciuto e trovarlo è compito dell'utente che vuole confermarlo.

Queste tre componenti entrano all'interno di una funzione SHA-256 che andrà a produrre un certo valore hash h che è richiesto essere inferiore ad una certa soglia e che abbia una sequenza iniziale di t zeri. L'utente che conferma il blocco, deve trovare il valore *nonce* tale che $h <$ soglia. Il *nonce* può essere trovato solo con una ricerca enumerativa (computazionalmente difficile), ma per verificarlo lo si può fare in maniera molto semplice (computazionalmente facile).

Il parametro t è calibrato dal sistema in base a quanto sono potenti e a quante sono le macchine che stanno cercando il *nonce* di modo che il tempo medio per trovarlo sia di 10 minuti.

18.3 Miner

È il nome degli utenti o nodi che validano le transazioni e aggiungono i blocchi alla block chain (**validazione tramite mining**). I miner provano continuamente a raggruppare transazioni per poi calcolarne il *nonce*, ovviamente sono tutti "in competizione" fra loro, quindi vince chi lo trova per primo. Chi riesce a trovarlo, lo diffonde in broadcast a tutti gli altri nodi.

A questo punto essi dovranno verificare se il *nonce* è valido

$$nonce + h_{i-1} + \text{transazioni} = 0_1 0_2 \dots 0_t \{0,1\}^{n-t}$$

e in caso affermativo dovranno esprimere il loro consenso, col quale "ammettono" che è stato aggiunto un nuovo blocco valido alla block chain, cercando di creare nuovi blocchi da attaccare.

Dato che il *nonce* non è semplicissimo da trovare, è difficile che due miner ne trovino due contemporaneamente. Tuttavia, qualora succeda, tipicamente i blocchi nati successivamente alla biforcazione, vengono attaccati al blocco che ha più transazioni e quella diventerà la block chain ufficiale. Anche questo processo si ripete sempre uguale per le poche volte che succede.

Nota bene: le uniche transazioni a cui è concesso di non specificare la transazione di input da cui prendere i BTC sono quelle aggiunte dai miner alla fine un blocco di transazioni.