# Indice

# Introduzione

La Crittografia studia i metodi per rendere inintelligibili (ossia *crittare*) messaggi in modo tale che siano comprensibili solo a persone designate (che li possono quindi *decrittare*) e che altri non li possano comprendere in tempo utile.

In questa tesi vengono presentati alcuni strumenti utilizzati dalla crittografia moderna, mettendo in luce le strutture algebriche e i risultati matematici sui quali si basano.

Per crittare messaggi si utilizza un *crittosistema*, che è definito come la collezione dei seguenti oggetti:

*a.* un insieme $PT$ formato da tutti i possibili testi in chiaro, ossia parole $m$ su un prefissato alfabeto $\Sigma$;

*b.* un insieme $CT$ dei crittotesti, ossia parole $c$ su un prefissato alfabeto $\Sigma'$);

*c.* un insieme delle chiavi $K$ e due algoritmi $D$ e $E$:

$$E : K \times PT \longrightarrow CT$$
$$D : K \times CT \longrightarrow PT$$

ogni chiave $k$ in $K$ determina un metodo di crittazione $E(k)$ e un metodo di decrittazione $D(k)$ tale che per ogni elemento $m \in PT$ sia $D(E(m)) = m$ (generalmente, come in questo caso, si sottointende la chiave $k$ e si indicano solo $E$ e $D$).

Coloro che utilizzano un crittosistema devono accordarsi sui dettagli degli oggetti descritti e utilizzarli come convenuto, ossia devono seguire un *protocollo*.

La ricerca di algoritmi utili a decrittare messaggi senza avere a disposizione $D(k)$ è detta *crittanalisi* e chi la pratica è detto *crittanalista*; si suppone

sempre che il crittanalista sia a conoscenza del crittosistema, in particolare, quando intercetta un messaggio, cercherà di calcolare $m$ o $k$ da $E(k, m)$ in tempo utile. Se riesce a presentare un metodo per questo calcolo che funzioni per ogni chiave $k$ e per ogni messaggio $m$, si dice che *il crittosistema è rotto.*

Il tempo necessario per crittare messaggi e per decrittarli senza avere a disposizione la chiave appropriata dipendono dalla scelta degli algoritmi e dell'insieme delle chiavi. Il primo capitolo della prima parte di questa tesi consiste in una breve rassegna di concetti introduttivi sulla Teoria della Complessità Computazionale, che ci permette di definire la velocità di un algoritmo. A tale scopo introduciamo la Macchina di Turing, assumendo che valga la Tesi di Church.

Definiamo poi la classe di complessità $P$ (ossia la classe dei problemi risolvibili in tempo polinomiale rispetto alla dimensione dell'input) e $NP$ (problemi risolvibili in tempo esponenziale, o infattibili).

Allo scopo di descrivere diversi sistemi crittografici, vengono richiamati alcuni risultati della Teoria dei Campi finiti e di Teoria dei Numeri, in particolare la *funzione $\varphi$ di Eulero*, che verrà utilizzata per il protocollo crittografico oggi più noto: l'RSA.

Inoltre, vengono introdotte le *curve ellittiche su un campo $K$*, che formano una famiglia di gruppi abeliani finiti. Questi sono utilizzati in crittografia sia per costruire un sistema crittografico basato sul logaritmo discreto, sia per rompere l'RSA, come vedremo in seguito.

Vengono infine presentati alcuni semplici concetti di Teoria dei Codici, visto che oggi la crittografia è utilizzata soprattutto in ambito informatico: daremo così le definizioni formali di messaggio e di canale di comunicazione.

La seconda parte tratta i crittosistemi cosiddetti classici, nei quali la conoscenza della chiave e dell'algoritmo utilizzati per crittare i messaggi, permettono di calcolare facilmente l'algorimo $D(k)$. Il più semplice di questi sistemi è il cosiddetto "sistema Caesar": ad ogni lettera del messaggio si associa il suo corrispondente numerico $\sigma$ nell'alfabeto $\Sigma = \{0, 1, \ldots, 26\}$ e per crittare si utilizza l'algoritmo di crittazione $E(k)$ (con $K = \{1, \ldots, 25\}$) che sostituisce $\sigma$ con

$$\sigma' = (\sigma + k) \pmod{26}.$$

E' ovvio che l'algoritmo di decrittazione $D(k)$ è :

$$D(k) : \sigma = (\sigma' - k) \pmod{26}$$

e che questo sistema risulti molto fragile.

Si passa quindi alla crittografia a chiave pubblica: in un sistema crittografico *a chiave pubblica* vengono resi noti sia $E$, sia la chiave $k \in K$ utilizzata; la relativa sicurezza del sistema si basa sul fatto che un crittanalista impiegherebbe troppo tempo per calcolare l'inverso di $E(k)$.

Gli esempi più noti sono basati sulla difficoltà computazionale di calcolare i fattori primi di un intero positivo "grande" (sistema RSA) e di calcolare il $\log_g a$ in un gruppo ciclico finito di ordine "grande". Nell'RSA la chiave di crittazione è data da $k = (n, e)$, dove $n$ è il prodotto di due primi $p$ e $q$, $1 < e < \varphi(n)$, $MCD(e, \varphi(n)) = 1$ e $\varphi$ indica la funzione di Eulero, ; se $m$ è il messaggio da inviare, l'algoritmo di crittazione è :

$$E(m) = m^e \pmod{n} = c;$$

e l'algoritmo di decrittazione $D$ è definito da:

$$D(c) = c^d \pmod{n},$$

dove $d = e^{-1} \pmod{\varphi(n)}$.

L'RSA sfrutta la difficoltà computazionale di trovare $e^{-1} \pmod{\varphi(n)}$ senza conoscere $\varphi(n)$ e di trovare $\varphi(n)$ senza conoscere $p$ e $q$.

Il problema del logaritmo discreto è il seguente:

> Sia $G$ un gruppo finito, dati $y, g \in G$, dire se esiste, e quindi calcolare, un $a \in \mathbb{N}$ tale che $g^a = y$.

Questo problema è al momento considerato tanto difficile quanto la fattorizzazione di un intero. In ordine cronologico, il primo sistema basato su di esso è lo scambio di chiavi di Diffie Hellman: se due persone devono scambiarsi la chiave $k$ da utilizzare per un sistema crittografico classico (normalmente più veloce di un sistema a chiave pubblica), possono utilizzare il seguente protocollo: scelgono un campo finito $F_q$ ed un suo generatore $g$ e li rendono pubblici; ciascuno sceglie un intero positivo (rispettivamente $a$ e $b$) e rendono pubblico, rispettivamente, $g^a$ e $g^b$; utilizzeranno, come chiave per il sistema crittografico classico l'elemento $g^{ab}$, che tutti e due possono calcolare: uno conosce $a$ perché lo ha scelto e $g^b$ perché è pubblico, l'altro conosce $b$ e $g^a$.

Viene poi descritto il protocollo di ElGamal, che consente lo scambio diretto di messaggi.

Oggi i sistemi basati sul logaritmo discreto si basano per lo più sul gruppo delle curve ellittiche, dal momento che la loro struttura algebrica, più debole rispetto a quella di campo, rende inutilizzabili alcuni algoritmi di crittanalisi.

Viene anche presentato il "problema dei pacchetti":

Dato un insieme $\{v_1, \ldots, v_k, V\}$ di interi, dire se esiste un insieme

$$\{\varepsilon_i | i = 1, \ldots, k;\ \varepsilon_i = 0, 1;\ \sum \varepsilon_i \cdot v_i = V\}$$

e presentare tale insieme,

il cui interesse risiede nel fatto che, malgrado questo problema sia di classe $NP$, i sistemi crittografici costruiti su di esso sono stati attaccati con successo.

Nella terza parte della tesi vengono discussi alcuni test di primalità, essenziali, per esempio, per la costruzione del sistema RSA.

Sono successivemente presentati gli algoritmi più noti per il calcolo del logaritmo discreto e per la fattorizzazione di un numero intero, utili alla crittanalisi.

In particolare si presenta un algoritmo per la fattorizzazione di un intero $n$, dovuto a Lenstra, che è basato su un algoritmo di Pollard che a sua volta sfrutta il Piccolo Teorema di Fermat in $\mathbb{Z}_n$. L'algoritmo di Lenstra fa uso delle Curve Ellittiche definite su $\mathbb{Z}_n$: se $n$ non è primo, allora $\mathbb{Z}_n$ non è un campo e la curva ellittica non ha la struttura di gruppo. Cominciando da un punto $P$ della curva, potrebbe non essere possibile trovare un multiplo $kP$ per un certo intero $k$; da questo $k$ si ricavano due fattori di $n$. Se si riesce a calcolare $kP$ anche per $k$ molto grandi, è preferibile cambiare curva ellittica e ricominciare il calcolo. Il grande numero di elementi della famiglia delle curve ellittiche consente di trovare una curva ed un suo punto tali che non è calcolabile $kP$ con buone probabilità .

Per completezza, nell'ultima parte della tesi, si sono voluti dare brevi cenni sui nuovi strumenti sui quali si basa la crittografia moderna, come la Teoria della Complessità Conoscitiva, e sui recenti protocolli crittografici utilizzati per firmare messaggi, per crittare messaggi in gruppo o per dimostrare la propria identità attraverso i canali informatici.

# Parte I

# Le basi

# 1

# Teoria della complessità computazionale

Dalla seconda metà degli anni '60, sotto la spinta dello sviluppo dei computer, sono state approfondite due importanti discipline tra loro correlate: la teoria della complessità computazionale (classificazione dei problemi conosciuti secondo la difficoltà) e l'analisi degli algoritmi (ricerca degli algoritmi migliori per la soluzione dei problemi in termini di spazio e tempo).

Solo dopo la nascita e lo sviluppo di queste due discipline è stata possibile la proposta di Whitfield Diffie e di Martin Hellman nel 1976 di un sistema crittografico a chiave pubblica, grazie al quale è possibile calcolare l'algoritmo di decrittazione, ma in un tempo troppo lungo perché il risultato possa essere utile (come illustrato nel capitolo 5).

In questo primo capitolo presentiamo le nozioni fondamentali e alcuni importanti risultati della teoria della complessità computazionale.

## 1.1 Algoritmi e macchine di Turing deterministiche

Agli inizi degli anni '20, un problema matematico molto diffuso era la definizione di "algoritmo arbitrario". Molti matematici del tempo proposero indipendentemente diversi modelli di computazione che risultarono tutti equivalenti tra loro ed in particolare ad uno di essi: la macchina di Turing. Questo risultato e le varie esperienze dell'epoca hanno portato alla Tesi di Church (non dimostrata):

*ogni algoritmo può essere visto come una macchina di Turing*

e ancora oggi essa è considerata l'esplicato formale della nozione intuitiva di algoritmo ([BCLR81]).

**Definizione 1.1.1.** Un *algoritmo* è una sequenza di operazioni (elementari) che permette di associare a dei dati di ingresso (input) un ben preciso risultato in uscita (output).

Un "buon algoritmo" è caratterizzato da:

*a.* ampiezza del campo di applicazione,

*b.* facilità di verifica delle ipotesi di applicazione,

*c.* propagazione limitata degli errori numerici (stabilità numerica), dovuti alle approssimazioni richieste dall'algoritmo,

*d.* richiesta contenuta di risorse (numero di operazioni, quantità di memoria, ecc.).

L'ultimo criterio è quello che individua la "complessità" di un algoritmo. In generale si valuta la richiesta di spazio e di tempo in funzione della dimensione dei dati in entrata e si parla di *complessità temporale* e di *complessità spaziale*, se si considerano lo spazio o il tempo massimi richiesti da input della stessa dimensione, o di *complessità prevista*, se è presa come media delle complessità richieste da input della stessa dimensione. La definizione dei concetti di complessità e di ottimalità di un algoritmo implica l'adozione di misure di costo che dipendono dal modello di computazione adottato. In generale, tale modello è rappresentato dalla macchina di Turing, che presentiamo con la Definizione 1.1.4; ma prima è necessario introdurre una classificazione dei problemi in base al tipo di risultato richiesto e alcune nozioni di Teoria dei Linguaggi:

**Definizione 1.1.2.** Dato un problema, esso è detto *decisionale* se richiede come risultato o un sì o un no; è detto *funzionale* se richiede risultati più elaborati.

Un caso particolare di problema funzionale è quello *computazionale*, che richiede l'esecuzione di operazioni algebriche.

In generale ogni problema funzionale può essere ridotto ad un caso particolare di tipo decisionale. Per esempio la ricerca dei fattori primi di un numero intero, può essere ridotta allo stabilire se il numero dato è primo (problema di primalità).

**Definizione 1.1.3.** Dato un insieme $A$, si definiscono:

$$A^n = A_1 \times \cdots \times A_n$$

per $A_i = A$ e

$$A^* = \bigcup_{n \in N} A^n.$$

Gli elementi di $A^*$ sono detti *stringhe di elementi di A*, e, per ogni $x \in A_1 \times \cdots \times A_n$, si definisce $n$ come *lunghezza della stringa* $x$ e la si denota con $|x|$.

Un qualunque insieme finito $A$ può essere detto *alfabeto*; in questo caso i suoi elementi sono detti *lettere* o *simboli*, gli elementi di $A^*$ sono detti *parole di A*, il sottoinsieme vuoto di $A$ è detto *parola nulla* o *spazio vuoto* e viene indicato con $\sqcup$, infine un qualsiasi sottoinsieme $L \subseteq (A - \sqcup)^*$ è detto *linguaggio su A*.

**Definizione 1.1.4.** Una *macchina di Turing (deterministica)* ad un nastro $M$ è l'insieme dei seguenti oggetti:

$\Sigma$ : un alfabeto le cui lettere vengono scritte su un nastro (vettore di $\Sigma^\infty$) e vengono indicate con $s_j$, dove $j$ è la posizione sul nastro. In $\Sigma$ ci devono essere gli elementi $\sqcup$ (spazio vuoto) e $\triangleright$ (simbolo iniziale); esempi di simboli sono l'insieme degli interi positivi $\{0, \ldots, 9\}$, e l'insieme $\{0, 1\}$. Il simbolo $\triangleright$ si pone sempre e solo in posizione $s_0$.

**un cursore** che legge e modifica gli elementi del nastro uno alla volta, secondo le istruzioni del programma (descritto più avanti);

$K$ : insieme degli stati della macchina. In esso devono essere sempre fissati un unico *stato iniziale* $k_0$ e un sottoinsieme di *stati finali* {h, sì, no} che sono tutti e soli gli stati che fanno fermare la macchina. Data una stringa input $x$ alla macchina, che viene scritta sul nastro, si denota con $M(x)$ il suo stato finale, che può essere sì nel caso che si proponga un problema decisionale con risposta positiva, no nel caso che si proponga un problema decisionale con risposta negativa o un problema computazionale non risolvibile, $h$ nel caso che si proponga un problema computazionale risolvibile. In questo ultimo caso si preferisce indicare con $M(x)$ una particolare sottostringa del nastro, delimitata da simboli convenzionali, che rappresenta il risultato della computazione. Se la macchina non si ferma si indica $M(x) = \nearrow$

$\delta$ : una funzione

$$K \times \Sigma \longrightarrow (K \times \Sigma \times \{\text{movimenti del nastro}\})$$
$$(k, s_j) \longmapsto (\bar{k}, \bar{s}_j, \text{movimento del nastro}),$$

detta *programma* o funzione di transizione. $s_j$ è il simbolo letto dal cursore, $\bar{k}$ è il nuovo stato della macchina, $\bar{s}_j$ è il nuovo simbolo che viene scritto sul nastro al posto di $s_j$ e i movimenti consentiti inducono la lettura o di $s_{j+1}$ o di $s_{j-1}$ o di $\bar{s}_j$ stesso. Il nuovo stato della macchina e il simbolo letto dopo il movimento del nastro costituiscono un nuovo argomento per il programma.

Questi oggetti devono inoltre soddisfare le seguenti proprietà:

- la macchina inizia sempre il lavoro dallo stato iniziale $k_0$ e dalla lettura di $s_0$;
- il simbolo $s_0 = \triangleright$ può essere sovrascritto unicamente da $\triangleright$.

**Esempio 1.1.1.** Sia $M$ una macchina di Turing deterministica con

$$K = \{k_0, k_1, k_2, k_3, h\}$$
$$\Sigma = \{0, 1, \triangleright, \sqcup\}$$

Si definisca il programma che introduce uno spazio vuoto tra la stringa input $x$ e $\triangleright$:

$$\begin{array}{lll}
\delta(k_0, 0) = (k_0, 0, \rightarrow) & \delta(k_0, 1) = (k_0, 1, \rightarrow) & \delta(k_0, \sqcup) = (k_1, \sqcup, \leftarrow) \\
\delta(k_0, \triangleright) = (k_0, \triangleright, \rightarrow) & \delta(k_1, 0) = (k_2, \sqcup, \rightarrow) & \delta(k_1, 1) = (k_3, \sqcup, \rightarrow) \\
\delta(k_1, \sqcup) = (q, \sqcup, -) & \delta(k_1, \triangleright) = (h, \triangleright, \rightarrow) & \delta(k_2, 0) = (k_0, 0, \leftarrow) \\
\delta(k_2, 1) = (k_0, 0, \leftarrow) & \delta(k_2, \sqcup) = (k_0, 0, \leftarrow) & \delta(k_2, \triangleright) = (h, \triangleright, \rightarrow) \\
\delta(k_3, 0) = (k_0, 1, \leftarrow) & \delta(k_3, 1) = (k_0, 1, \leftarrow) & \delta(k_3, \sqcup) = (k_0, 1, \leftarrow) \\
\delta(k_3, \triangleright) = (h, \triangleright, \rightarrow) & &
\end{array}$$

dove $\leftarrow$ e $\rightarrow$ indicano che il simbolo successivo letto dal cursore è $s_{j-1}$ o $s_{j+1}$.

Sia $x = 010$, quindi all'inizio del lavoro il nastro è rappresentato dal vettore

$$(\underline{\triangleright}, 0, 1, 0),$$

dove è sottolineato il simbolo sul cursore. Lo stato iniziale è $k_0$ e, pertanto, la prima azione della macchina, come descritto dal programma, è il risultato della

$$\delta(k_0, \triangleright) = (k_0, \triangleright, \rightarrow)$$

che riscrive $\triangleright$ su $\triangleright$, lascia lo stato invariato e sposta il cursore a destra:

$$(\triangleright, \underline{0}, 1, 0),$$

con stato $k_0$ e l'azione successiva sarà

$$\delta(k_0, 0) = (k_0, 0, \rightarrow)$$

che porta alla coppia stato - nastro:

$$k_0, (\triangleright, 0, \underline{1}, 0).$$

Proseguendo così si hanno le seguenti coppie stato - nastro:

| | | |
|---|---|---|
| $k_0$,$(\triangleright, 0, 1, \underline{0})$ | $k_0$,$(\triangleright, 0, 1, 0, \underline{\sqcup})$ | $k_1$,$(\triangleright, 0, 1, \underline{0}, \sqcup)$ |
| $k_2$,$(\triangleright, 0, 1, \sqcup, \underline{\sqcup})$ | $k_0$,$(\triangleright, 0, 1, \underline{\sqcup}, 0)$ | $k_1$,$(\triangleright, 0, \underline{1}, \sqcup, 0)$ |
| $k_3$,$(\triangleright, 0, \sqcup, \underline{\sqcup}, 0)$ | $k_0$,$(\triangleright, 0, \underline{\sqcup}, 1, 0)$ | $k_1$,$(\triangleright, \underline{0}, \sqcup, 1, 0)$ |
| $k_2$,$(\triangleright, \sqcup, \underline{\sqcup}, 1, 0)$ | $k_0$,$(\triangleright, \underline{\sqcup}, 0, 1, 0)$ | $k_1$,$(\underline{\triangleright}, \sqcup, 0, 1, 0)$ |
| $h$,$(\triangleright, \underline{\sqcup}, 0, 1, 0)$ | | |

dove il risultato è quella porzione di nastro compresa tra $\triangleright$ ed il secondo $\sqcup$ (i simboli convenzionali descritti nella definizione 1.1.4) e, quindi,

$$M(010) = (\sqcup, 0, 1, 0)$$

*Osservazione.* Quella appena descritta è una macchina di Turing deterministica, che si differenzia da quelle non deterministiche, che vedremo in seguito (paragrafo 1.4), perché ogni sua azione è univocamente determinata dallo stato e dal simbolo letto dal cursore.

Dato un problema decisionale, quindi, lo si trasforma in una stringa $x$ di elementi di un certo alfabeto $\Sigma$, che viene valutata da un algoritmo (ossia da una macchina di Turing, se si ritiene valida la Tesi di Church): se la stringa appartiene ad un certo *linguaggio* allora la risposta sarà positiva. Poiché le macchine di Turing sono oggi rappresentate dai computer, si preferisce usare la notazione binaria, ma è dimostrato [SA91] che scelte diverse di $\Sigma$ non influiscono sulla velocità del procedimento in modo significativo.

**Definizione 1.1.5.** Dato un linguaggio $L$ e una macchina di Turing $M$, si dice che $M$ *decide* $L$ se

$$M(x) = \text{sì} \quad \forall x \in L \quad \text{e} \quad M(x) = \text{no} \quad \forall x \notin L;$$

in questo caso si dice anche che $L$ *è un linguaggio ricorsivo.*

Una macchina di Turing $M$ *accetta il linguaggio* $L$ se

$$M(x) = \text{sì} \quad \forall x \in L \quad \text{e} \quad M(x) = \nearrow \forall x \notin L;$$

in questo caso si dice anche che $L$ *è un linguaggio ricorsivamente numerabile.*

Nel caso di problemi computazionali valgono le stesse definizioni, sostituendo a sì il simbolo h. In questo caso, se $M$ decide $L$, si dice anche che $M$ *computa* $L$, ed $L$ è detto *linguaggio computabile.*

E' pressoché immediata la dimostrazione del seguente teorema:

**Teorema 1.1.1.** *$L$ ricorsivo $\Rightarrow$ $L$ ricorsivamente numerabile.*

**Definizione 1.1.6.** Data una funzione

$$f : (\Sigma - \sqcup)^* \longrightarrow \Sigma^*,$$

rappresentatrice di un problema computazionale, si dice che *la macchina di Turing $M$ computa $f$* se $\forall x \in (\Sigma - \sqcup)^*$ si ha $M(x) = f(x)$; $f$ è detta funzione ricorsiva.

## 1.2 Tempo e classi di complessità

Per ogni problema ci possono essere più algoritmi e la ricerca si propone di stabilire quale di questi possa essere il più veloce. In questo paragrafo definiremo il concetto di velocità per gli algoritmi, e questo ci permetterà di suddividerli in quelle che definiremo "classi di complessità".

**Definizione 1.2.1.** In una macchina di Turing $M$ si introduce la scrittura $(q, u, v)$, detta *configurazione ad un certo istante*, dove $q$ è lo stato, $u$ la parte di nastro a sinistra del cursore insieme al simbolo letto (e non ancora modificato) dal cursore, $v$ la parte di nastro a destra del cursore.

Si dice che *la configurazione* $(q, u, v)$ *porta alla configurazione* $(q', u', v')$ se, dopo un numero finito $k$ di passi, $M$ passa dalla prima alla seconda configurazione; si può scrivere:

$$(q, u, v) \xrightarrow{M^k} (q', u', v').$$

**Esempio 1.2.1.** Nell'esempio precedente veniva scritto tutto il nastro, con il simbolo letto dal cursore sottolineato, mentre con questa notazione le prime configurazioni sono indicate come:

$$(k_0, \triangleright, (0, 1, 0)) \quad (k_0, (\triangleright, 0), (1, 0)) \quad (k_0, (\triangleright, 0, 1), 0),$$

per arrivare alla configurazione finale

$$(h, (\triangleright, \sqcup), (0, 1, 0)).$$

**Definizione 1.2.2.** Se la configurazione iniziale $(s, \triangleright, x)$ porta alla configurazione finale $(H, u', v')$, dove $x$ è la stringa in entrata e $H \in \{\text{sì, no, h}\}$, il minimo $k$ per cui

$$(s, \triangleright, x) \xrightarrow{M^k} (H, u', v')$$

è detto *tempo richiesto da $M$ per l'input $x$.*

Come già accennato nel paragrafo precedente, rappresentiamo la prestazione di una macchina di Turing in termini di spazio e di tempo in funzione della lunghezza della stringa in entrata:

**Definizione 1.2.3.** Si dice che *$M$ lavora entro un tempo $f(n)$* se per ogni stringa $x$ in entrata con lunghezza $n$, il tempo massimo impiegato da $M$ è $f(n)$.

Dato un linguaggio $L \subseteq (\Sigma - \sqcup)^*$, si dice che

$$L \in \text{TIME}(f(n))$$

se esiste una macchina di Turing che decida tutti gli elementi di $L$ entro il tempo $f(n)$. TIME $(f(n))$ è una classe di complessità, dove:

**Definizione 1.2.4.** Si dice *classe di complessità* una collezione di linguaggi che possono essere decisi o computati lasciando limitato un parametro di costo (tempo, spazio, altro); il parametro e il limite definiscono la classe.

Devono essere soddisfatte le seguenti proprietà:

*a.* deve essere fissato un "modello di computazione", che, nel seguito, sottintenderemo essere una macchina di Turing ad un nastro;

*b.* deve essere fissato un "modo di computazione", che può essere, a titolo di esempio, deterministico o non deterministico;

*c.* deve essere fissata la risorsa da limitare (spazio, tempo, ecc.);

*d.* il limite deve essere dato da una funzione $f : \mathbb{N} \longrightarrow \mathbb{N}$ monotonamente non decrescente, il cui argomento è la dimensione dell'input.

Ricordiamo la seguente definizione:

**Definizione 1.2.5.** Siano $f, g : \mathbb{N}^r \longrightarrow \mathbb{R}$ due funzioni, $x \in \mathbb{N}^r$; si indica

$$f = O(g) \text{ per } n \to x$$

se

$$\exists c \in \mathbb{R} \quad \text{t.c.} \quad \limsup_{n \to x} \frac{f(n)}{g(n)} = c.$$

Normalmente non si indica il limite $x$ se $x = \infty^r$.

Questo ci consente di considerare le classi $\text{TIME}(O(f(n)))$:

**Definizione 1.2.6.** Si dice che $L$ è un *linguaggio decidibile polinomialmente* se esiste un $k \in \mathbb{N}$ tale che $L \in \text{TIME}(O(n^k))$. Si definisce

$$P = \bigcup_{k \in \mathbb{N}} \text{TIME}(O(n^k)).$$

Un problema è detto *intrattabile (o impossibile) computazionalmente* se non è in $P$.

### Altre macchine di Turing deterministiche

Secondo Church, una macchina di Turing può essere vista come un algoritmo, ma ci si potrebbe chiedere come un meccanismo così semplice possa rappresentare anche algoritmi piuttosto complicati. Si potrebbe introdurre la seguente:

**Definizione 1.2.7.** Si dice *macchina di Turing deterministica a k nastri (o a nastri multipli)*, l'insieme dei dati $K$, $s$, $\Sigma$ e $\delta$, dove

$$\delta : K \times \Sigma^k \longrightarrow K \times (\Sigma \times \{\text{movimenti}\})^k$$

con le proprietà di avere $\triangleright$ sempre e solo all'inizio di ogni nastro e di riscrivere $\triangleright$ alla lettura di $\triangleright$ su ogni nastro. Si ha la scrittura

$$\delta(q, s_1, \dots, s_k) = (p, r_1, M_1, \dots, r_k, M_k).$$

A questo punto potrebbe sembrare più facile vedere come una macchina di questo tipo possa rappresentare un algoritmo: per esempio legge i dati in entrata da un nastro, li elabora attraverso il programma e gli altri nastri, scrive il risultato sull'ultimo nastro. Potrebbe anche sembrare più conveniente (sia in termini di tempo, sia per quanto riguarda la quantità dei problemi trattabili) il suo utilizzo rispetto ad una macchina di Turing deterministica a un nastro, ma si dimostra il seguente teorema:

**Teorema 1.2.1.** *Data una macchina di Turing a $k$ nastri $M$ che decide il linguaggio $L$ in un tempo $f(n)$, si può costruire una macchina $M'$ ad un solo nastro che lavora in tempo $O(f(n)^2)$ e tale che $\forall x \in L$ è $M(x) = M'(x)$.*

Nel corso degli anni, sono state costruite macchine apparentemente ancora più potenti, come le macchine RAM (Random Access Machines, sulle quali è basato il funzionamento dei computer), ma neanche queste rappresentano un reale miglioramento della macchina di Turing ad un nastro in termini di aumento dei linguaggi decidibili o di aumento delle prestazioni più che polinomialmente, come dimostrato dal seguente teorema:

**Teorema 1.2.2.** *Data una macchina RAM $\bar{M}$ che decide il linguaggio $L$ in un tempo $f(n)$, si può costruire una macchina di Turing deterministica $M'$ ad un solo nastro che lavora in tempo $O(f(n)^3)$ e tale che $\forall x \in L$ è $\bar{M}(x) = M'(x)$.*

## 1.3 Spazio

Le risorse più importanti che vengono utilizzate dalle macchine di Turing, come ci insegna l'esperienza quotidiana con i computer, sono il tempo e lo spazio. In questo paragrafo definiremo formalmente questa seconda risorsa, che ci permetterà di definire un'altra classe di complessità. Per fare questo dobbiamo presentare un altro tipo di macchina di Turing:

**Definizione 1.3.1.** Una *macchina di Turing con input ed output* è una macchina di Turing a $k$ nastri, tale che, se indichiamo con

$$\delta(q, s_1, \dots, s_k) = (p, r_1, M_1, \dots, r_k, M_k)$$

il suo programma, abbia le proprietà:

- $r_1 = s_1$ (quindi il primo nastro è solo di lettura);
- il $k$-esimo nastro non si muove mai a sinistra (ossia non torna mai indietro, quindi è solo di scrittura);

Si supponga quindi di avere una macchina di Turing con input ed output a $k$ nastri, se la sua configurazione finale (da input $x$) è data da

$$(H, u_1, v_1, \dots, u_k, v_k),$$

si dice che *lo spazio richiesto da $M$ per il dato $x$* è

$$\sum_{i=1}^{k} |u_i||v_i|.$$

Data una funzione $f : \mathbb{N} \longrightarrow \mathbb{N}$, si dice che $M$ opera in uno *spazio limite f(n)* se, per ogni input $x$ di lunghezza $n$, $M$ richiede spazio minore o uguale a $f(n)$.

Si dice che il linguaggio $L$ è nella classe spaziale di complessità

$$\text{SPACE}(f(n))$$

se esiste una macchina di Turing con input ed output che decide $L$ ed opera in uno spazio limitato da $f(n)$.

Si possono introdurre le classi di complessità spaziale del tipo

$$\text{SPACE}(O(f(n))).$$

## 1.4 Macchine di Turing non deterministiche

**Definizione 1.4.1.** Una *macchina di Turing non deterministica* $N$ è l'insieme dei dati:

$\Sigma$**, un cursore,** $K$ definiti e con le stesse proprietà della Definizione 1.1.4;

**una relazione** $\Delta$ (e non più la funzione $\delta$), con

$$\Delta \subseteq (K \times \Sigma) \times [K \times \Sigma \times \{\text{movimenti del nastro}\}]$$

dove $K \times \Sigma$ è la lettura dello stato e del nastro, e

$$K \times \Sigma \times \{\text{movimenti del nastro}\}$$

è l'insieme dei "risultati", non univocamente determinati dalla coppia (stato, simbolo) letta.

La si può vedere come una macchina che "tira ad indovinare", ossia che può scegliere, di volta in volta, un qualche "risultato" diverso alla lettura di una determinata coppia $(q, s)$.

**Definizione 1.4.2.** La differenza tra queste e le macchine di Turing è la nozione che si dà per "risolvere il problema": si dice che $N$ *decide* $L$ *(linguaggio)* se per ogni $x \in L$ esistono un intero positivo $k$ e due stringhe $u, v \in L$ tali che:

$$(s, \triangleright, x) \xrightarrow{N^k} (\text{sì}, u, v)$$

**Definizione 1.4.3.** Sia data una funzione $f : \mathbb{N} \longrightarrow \mathbb{N}$; si dice che $N$ *decide* $L$ *in tempo* $f(n)$ se per ogni $x \in \Sigma^*$ di lunghezza $n$ esistono un intero positivo $k \leq f(n)$ e due stringhe $u, v \in L$ tali che

$$(s, \triangleright, x) \xrightarrow{N^k} (\text{sì}, u, v).$$

Si può vedere $f(n)$ come la lunghezza del cammino più lungo necessario per decidere $x$, oppure come il limite di tempo massimo per verificare se è stata "indovinata" o meno una soluzione.

In questo caso si denota

$$L \in \text{NTIME}(f(n))$$

e si definisce

$$NP = \bigcup_{k \in N} \text{NTIME}(O(n^k)).$$

*Osservazione.* $P \subseteq NP$, infatti $P$ è un caso particolare di $NP$, dove la relazione $\Delta$ è una funzione.

**Esempio 1.4.1.** La fattorizzazione di un intero è un problema $NP$: uno "indovina" i fattori primi, ne calcola il prodotto e verifica il risultato. Non si sa se è un problema di tipo $P$.

Una delle questioni più importanti della Teoria della Complessità è:

$$P = NP?$$

Si suppone [Pa95] che l'uguaglianza non valga; in ogni caso il seguente teorema stabilisce una relazione tra le classi TIME $(f(n))$ e NTIME $(f(n))$:

**Teorema 1.4.1.** *E' possibile trasformare un algoritmo non deterministico in uno deterministico, però si ha un aumento del tempo in termini esponenziali, ossia: se* $L$ *è deciso da una macchina di Turing* $N$ *non deterministica in tempo* $f(n)$*, esiste una macchina di Turing* $M$ *deterministica che lo decide in tempo* $O(c^{2f(n)})$ *con* $c$ *costante dipendente da* $L$ *e da* $N$*.*

Diamo infine una bella caratterizzazione dei problemi $NP$ (per la dimostrazione si veda [Pa95]) che risulterà utile in seguito:

**Teorema 1.4.2.** *Sia $L$ linguaggio di $\Sigma^*$. $L \in NP$ se e solo se esiste una relazione $R$ in $\Sigma^* \times \Sigma^*$ tale che:*

a. *esista una macchina di Turing deterministica $M$ che decida il linguaggio:*

$$\{x, y \mid (x, y) \in R\}$$

b. *per ogni coppia $(x, y) \in R$ esista un intero positivo $k$ tale che $|y| \leq |x|^k$, dove $|x|$ è la lunghezza della stringa $x$;*

c. $L = \{x \in \Sigma^* \mid (x, y) \in R \quad \text{per un certo} \quad y \in \Sigma^*\}$.

Questa caratterizzazione afferma che per ogni problema decisionale in $NP$, se la risposta all'input $x$ è sì, esiste almeno un "testimone polinomiale" o "certificato" $y$ della positività di $x$. Si può anche non saper calcolare $y$, ma la sua esistenza è garantita per $M(x) =$ sì.

Dato questo teorema è possibile introdurre una nuova classe di complessità :

**Definizione 1.4.4.** La classe $coNP$ è la classe i cui problemi decisionali hanno un "certificato" in caso di risposta negativa.

**Esempio 1.4.2.** Il problema della primalità è in $NP \cap coNP$, infatti è in $coNP$ perché , per ogni intero non primo, ogni suo divisore proprio rappresenta il "certificato", ed è in $NP$ perché il suo certificato è dato dall'intero $r$ definito dal seguente teorema:

**Teorema 1.4.3.** *$p \in \mathbb{N}$ è primo se e solo se esiste un intero positivo $r$, $1 \leq r \leq p$ tale che*

$$r^{p-1} \equiv 1 \pmod{p} \quad e \quad r^{(p-1)/q} \not\equiv 1 \pmod{p}$$

*per ogni divisore primo $q$ di $p - 1$.*

## 1.5 Algoritmi probabilistici

Alcuni degli algoritmi più efficaci per risolvere il problema della primalità, come vedremo nel paragrafo 6.1, sono di tipo *probabilistico* o *casuale*, che definiamo qui di seguito.

**Definizione 1.5.1.** Dato un linguaggio $L$, si dice che è *decidibile casualmente in tempo polinomiale* se esiste un polinomio $p$ ed una macchina di Turing deterministica polinomiale che computi, per ogni parola $x$ di $L$ ed ogni possibile "certificato" $y$ di lunghezza $p(|x|)$, un valore $v(x, y) \in \{0, 1\}$ tale che:

- se $x \notin L$ si ha $v(x, y) = 0$ per ogni $y$ ;
- se $x \in L$ si ha $v(x, y) = 1$ per almeno la metà dei "certificati" $y$.

Si indica con $RP$ la classe dei problemi che sono decidibili casualmente in tempo polinomiale. Intuitivamente, se $x \notin L$ non passerà il test al quale è sottoposta, che generalmente consiste nella verifica di una condizione necessaria che, per la seconda proprietà, se è in $L$, viene soddisfatta per almeno metà delle prove.

*Osservazione.* $P \subseteq RP \subseteq NP$. Infatti, in $P$ la funzione $v$ è indipendente da $y$ e in $NP$ si richiede soltanto che $v(x, y) = 1$ per almeno un $y$.

## 1.6 Completezza

In questo paragrafo introdurremo la nozione di linguaggi completi, caso particolare dei quali sono i linguaggi $NP$-completi, che rappresentano i problemi computazionalmente intrattabili.

Per una definizione formale di completezza è necessaria la definizione di linguaggio riducibile, che, nella sua forma più comune, ha bisogno del seguente lemma, che stabilisce una relazione tra la richiesta di spazio e la velocità di un algoritmo:

**Lemma 1.6.1.** *Per ogni trasformazione tra stringhe computabile da una macchina di Turing deterministica $M$ in spazio $O(\log n)$ ($n$ lunghezza dell'input), esiste un $k \in \mathbb{N}$, tale che $M$ si fermi dopo un numero massimo di passi $O(n^k)$.*

**Definizione 1.6.1.** Si dice che il linguaggio $L_1$ è riducibile al linguaggio $L_2$ se esiste $R$ funzione da stringhe a stringhe, $R$ computabile da una macchina di Turing deterministica in spazio $O(\log n)$ ($n$ lunghezza della stringa $x \in L_1$), tale che $x \in L_1$ se e solo se $R(x) \in L_2$. $R$ è detta *riduzione da $L_1$ a $L_2$*.

In sostanza, dati due problemi A e B, si dice che B si riduce ad A se esiste una trasformazione $R$ tale che:

*a.* per ogni possibile input $x$ di B, $R(x)$ sia un possibile input di A;

*b.* il risultato del problema A con input $R(x)$ sia lo stesso del risultato del problema B con input $x$;

*c.* il calcolo di $R$ deve essere di classe temporale TIME ($O(n^k)$) (quindi polinomiale) se $n$ è la lunghezza di $x$, come illustrato dal Lemma 1.6.1.

**Definizione 1.6.2.** Sia C una classe di complessità, sia $L$ un linguaggio di C. Si dice che $L$ *è C-completo* se ogni linguaggio $L'$ di C può essere ridotto a $L$.

**Definizione 1.6.3.** Una classe di complessità C si dice *chiusa rispetto alle riduzioni* se per ogni coppia di linguaggi $L_1$ e $L_2$ con $L_2 \in$ C, $L_1$ riducibile a $L_2$ implica $L_1 \in$ C.

Da queste definizioni si ha il seguente teorema:

**Teorema 1.6.1.** *Se C e C' sono classi di complessità chiuse rispetto alle riduzioni ed esiste un linguaggio $L \in C \cap C'$ completo sia per C che per C', allora C=C'.*

Come già detto nel paragrafo 1.4, uno dei problemi aperti della Teoria della Complessità è se $P = NP$, che viene riproposto come:

esiste un problema $NP$-completo che stia in $P$?

Normalmente si crede che $P \neq NP$ e che, pertanto, i problemi $NP$-completi siano intrattabili, dal momento in cui un problema $NP$-completo non è di classe $P$.

La pratica comune dice che ogni problema computazionale, se generalizzato abbastanza, diventa un problema $NP$-completo o non classificabile, e ogni problema ha un caso particolare in $P$ ([SA96]).

## 1.7 Calcolo della velocità di un algoritmo (il tempo per fare la matematica)

La crittografia a chiave pubblica, come vedremo nel capitolo 5, si basa sul fatto che è possibile trovare la chiave di decifrazione usata, ma in un tempo troppo lungo perché il risultato possa essere utile. Ecco perché è stata introdotta la teoria della complessità: dato un sistema crittografico a

chiave pubblica, vedere se esiste un algoritmo per romperlo e quanto tempo impiega a trovare il risultato.

Abbiamo sin qui visto che la Teoria della Complessità considera lo spazio o il tempo massimi richiesti da input della stessa dimensione, ossia la "peggiore prestazione". Per alcuni problemi, però, la maggior parte dei casi risulta essere molto meno laboriosa del caso peggiore; uno studio più approfondito della "complessità media" richiederebbe una teoria sulla distribuzione degli input di un dato problema, che al momento non sembra sufficientemente sviluppata [Pa95]. In crittografia, comunque, la complessità media è più importante della peggiore prestazione: un sistema crittografico a chiave, per essere facilmente utilizzabile, deve risultare sicuro per la maggioranza delle chiavi possibili e non per qualche raro caso.

La Teoria della Complessità si basa sull'uso degli $O$, perdendo così i coefficienti dell'argomento, ma la continua crescita delle velocità dei computer non rende necessaria la costruzione di una teoria alternativa [GO97].

Noi saremo interessati a studiare classi di complessità relative alla risorsa "tempo" e adotteremo un'unità di misura del tempo più utile ai nostri scopi rispetto al numero di passi di una macchina di Turing, ossia il numero di operazioni binarie necessarie per ottenere un risultato dall'algoritmo in funzione del numero di cifre binarie dell'input.

Sia $n$ un intero che deve essere riscritto in base $b$, ($n$ e $b$ sono da intendersi in base decimale) allora il numero di cifre di $n$ in base $b$ è dato dalla

$$\#\text{cifre} = [\log_b n] + 1 = \left[\frac{\log n}{\log b}\right] + 1 = O(\log n),$$

e questo risultato è indipendente dalla base in cui è scritto in numero.

**Definizione 1.7.1.** La somma di due bit (ossia di due elementi di $\mathbb{Z}_2$) viene detta *operazione binaria* (bit operation).

**Esempio 1.7.1 (Somma di due interi).** La somma di due numeri interi di $k$ bit richiede $k$ operazioni binarie.

La somma di due interi positivi $n > m$ in base dieci è quindi un problema computazionale di classe TIME ($O(\log n)$).

**Esempio 1.7.2 (Moltiplicazione di interi).** La moltiplicazione di un intero di $k$ cifre per un intero di $j$ cifre, con $j \leq k$, con l'algoritmo elementare, richiede al massimo $j$ somme di interi che hanno al massimo $k + j$ bit; in definitiva questo algoritmo richiede $(j(k + j)) \leq 2kj = O(kj)$ operazioni binarie.

Se $m$ ed $n$ sono due interi positivi in base decimale, la loro moltiplicazione richiede $2(\log n)(\log m) = O((\log n)(\log m))$ operazioni binarie.

E' possibile lavorare anche con frazioni; in questo caso la scrittura del numero in base $b$ sarà $(d_{k-1}, \ldots, d_0, d_{-1}, \ldots, d_{-j})$. I risultati fin qui ottenuti con i numeri interi sono uguali a quelli ottenibili con numeri frazionari.

**Esempio 1.7.3 (Massimo comun divisore di due interi).** Dati due interi $a$ e $b$, con $a > b$, allora:

$$\text{TIME}(MCD(a,b) \text{ utilizzando l'algoritmo Euclideo}) = O(\log^3 a)$$

# 2

# Basi algebriche

La crittografia moderna si basa su alcuni risultati di Teoria dei Numeri, quindi, in questo capitolo, forniremo alcune definizioni e risultati utili per il seguito.

## 2.1 Alcune definizioni

Come vedremo, per la costruzione del sistema crittografico a chiave pubblica oggi più diffuso, l'RSA, serve la funzione di Eulero:

**Definizione 2.1.1.** Sia $n \in \mathbb{N}$, si definisce

$$\varphi(n) = |\{0 \leq b < n \text{ t.c. } MCD(b, n) = 1\}|$$

e viene detta *funzione $\varphi$ di Eulero.*

**Teorema 2.1.1.** *La funzione $\varphi$ di Eulero è moltiplicativa, quindi $\forall m, n \in \mathbb{N}$ si ha, per $MCD(m, n) = 1$,*

$$\varphi(mn) = \varphi(m) \cdot \varphi(n).$$

*Dimostrazione.* Sia $j \in \mathbb{N}$, con $0 < j \leq mn - 1$ tale che $MCD(j, mn) = 1$. Siano

$$j_1 \equiv j \pmod{m} \quad \text{e} \quad j_2 \equiv j \pmod{n}.$$

tali che $0 \leq j_1 < m$ e $0 \leq j_2 < n$. Per il Teorema Cinese del Resto esiste un unico $j$ compreso tra 0 e $mn - 1$ tale che

$$j \equiv j_1 \pmod{m} \quad \text{e} \quad j \equiv j_2 \pmod{n}.$$

Ora, $j$ è primo con $mn$ se e solo se è primo con $m$ e con $n$, se e solo se $j_1$ è primo con $m$ e $j_2$ è primo con $n$. Quindi un tale $j$ deve essere in corrispondenza biunivoca con una coppia $(j_1, j_2)$ tale che:

$$\begin{aligned} 0 \leq j_1 < m \qquad & MCD(j_1, m) = 1 \\ 0 \leq j_2 < n \qquad & MCD(j_2, n) = 1, \end{aligned}$$

ed il numero di tali $j_1$ è $\varphi(m)$, mentre il numero dei $j_2$ è $\varphi(n)$; pertanto il numero delle coppie $(j_1, j_2)$ cercate è $\varphi(m)\varphi(n)$. □

*Osservazione.* Si osserva che $\varphi(1) = 1$, $\varphi(p) = p - 1$ per $p$ primo e

$$\varphi(p^\alpha) = p^\alpha - p^{\alpha-1} = p^\alpha \left(1 - \frac{1}{p}\right)$$

per ogni primo $p$ (dal momento in cui l'unico primo che divide $p^\alpha$ è $p$ e ci sono esattamente $p^{\alpha-1}$ suoi multipli compresi tra 0 e $p^\alpha - 1$).

Per ogni intero $n$ con decomposizione in primi $n = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$, con $p_i \neq p_j$, si ha

$$\varphi(n) = \prod_{i=1}^{k} \varphi(p_i^{\alpha_i}) = n \prod_{p|n} \left(1 - \frac{1}{p}\right).$$

**Teorema 2.1.2 (di Eulero).** *Siano $a, b \in \mathbb{N}$; se $MCD(a, b) = 1$ si ha:*

$$a^{\varphi(b)} \equiv 1 \pmod{b}.$$

*Dimostrazione.* Si consideri il caso in cui $b$ sia una potenza di primo: $b = p^\alpha$ e si dimostri per induzione su $\alpha$: per $\alpha = 1$ vale il Piccolo Teorema di Fermat (Teorema 2.3.1); sia $\alpha \geq 2$ e si supponga che la formula valga per $p^{\alpha-1}$, da cui:

$$a^{\varphi(p^{\alpha-1})} = a^{p^{\alpha-1} - p^{\alpha-2}} \equiv 1 \pmod{b} \equiv 1 \pmod{p^{\alpha-1}}$$

per cui $\exists c \in \mathbb{Z}$ tale che

$$a^{p^{\alpha-1} - p^{\alpha-2}} = 1 + c \cdot p^{\alpha-1};$$

elevando i due membri alla $p$ ed utilizzando il fatto che i coefficienti binomiali nell'espansione di $(1+x)^p$ sono tutti divisibili per $p$ (eccetto il primo e l'ultimo addendo che risulta essere $x^p$), si ha che

$$a^{p^\alpha - p^{\alpha-1}} = (1 + c \cdot p^{\alpha-1})^p$$

è uguale ad 1 sommato a multipli di $p^\alpha$, quindi

$$a^{\varphi(p^\alpha)} = a^{p^\alpha - p^{\alpha-1}} \equiv 1 \pmod{p^\alpha}.$$

Nel caso generale, per il Teorema 2.1.1, basta moltiplicare l'equazione

$$a^{\varphi(p^\alpha)} \equiv 1 \pmod{p^\alpha}$$

per la potenza appropriata per verificare che

$$a^{\varphi(b)} \equiv 1 \pmod{p^\alpha};$$

poiché ciò è vero per ogni $p^\alpha$ che divide $b$ e ogni $p^\alpha$ non ha alcun divisore proprio con gli altri fattori primi di $b$, per le proprietà elementari delle congruenze si ha la tesi. □

**Corollario 2.1.1.** *Siano $p, m \in \mathbb{N}$, con $p$ primo, allora*

$$m^{p-1} \equiv 1 \pmod{p}.$$

L'RSA si basa sulla difficoltà di fattorizzare i numeri interi; ma, se questi hanno fattori primi "piccoli", esistono algoritmi che permettono la soluzione di questo problema in poco tempo (capitolo 6.2); a tale proposito diamo la seguente definizione:

**Definizione 2.1.2.** Si dice che $n \in \mathbb{N}$ è *liscio* se i suoi fattori primi sono "piccoli".

Non risulterà superfluo, infine, ricordare la seguente notazione:

**Definizione 2.1.3.** Dato $x \in \mathbb{R}$, si indica con $[x]$ la sua parte intera, e $[x] \leq x$.

## 2.2 Campi finiti

Indicheremo con $F_q$ un campo finito di $q$ elementi e con $F_q^*$ il suo gruppo moltiplicativo di ordine $q-1$; indicheremo con $F[x]$ l'anello dei polinomi su un campo $F$, e con $F[x]^n$ il suo sottoinsieme

$$F[x]^n = \{f(x) \in F[x] | \deg f(x) \leq n\}.$$

Se $F_q$ è un campo, allora tutti i suoi elementi hanno come caratteristica lo stesso numero primo $p$, ed esiste un intero positivo $f$ tale che $q = p^f$.

Viceversa, per ogni primo $p$ e per ogni intero positivo $n$, esiste, a meno di isomorfismi, un unico campo finito di ordine $p^n$.

Dato un primo $p \in \mathbb{N}$ ed un intero positivo $n$, si può costruire esplicitamente il campo $F_q$ con $q = p^n$: sia $F_p \simeq \mathbb{Z}_p$ e sia

$$f(x) \in F_p[x]$$

un polinomio irriducibile di ordine $n$ (si dimostra che, dato un campo finito $F$, per ogni $n \in \mathbb{N}$, esiste un polinomio irriducibile di grado $n$ in $F[x]$). Indichiamo con $(f(x))$ l'ideale di $F_p[x]$ generato da $f(x)$, che si dimostra essere massimale, e pertanto l'anello quoziente

$$\frac{F_p[x]}{(f(x))}$$

è un campo. Gli elementi di questo campo sono classi di equivalenza con rappresentanti tutti e soli i polinomi di $F_p[x]$ di grado minore o uguale a $n-1$, e pertanto questo campo ha ordine $p^n$ e

$$\frac{F_p[x]}{(f(x))} \simeq \{a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 \text{ tale che } a_i \in F_p\} \simeq F_q.$$

Da ciò si deduce che ogni elemento di $F_q = F_{p^n}$ può essere visto come polinomio di grado $n-1$ definito su $F_p$.

**Teorema 2.2.1.** *Ogni campo finito $F_q$ ha un elemento di ordine $q-1$, detto* generatore *(e, quindi, il gruppo moltiplicativo $F_q^*$ è ciclico). Se $g$ è un generatore di $F_q$ allora $g^j$ è generatore di $F_q$ se e solo se $MCD(j, q-1) = 1$; in particolare ci sono $\varphi(q-1)$ generatori di $F_q$.*

Per dimostrare questo Teorema è necessario il seguente Lemma:

**Lemma 2.2.1.** *Per ogni intero positivo $n$ si ha:*

$$\sum_{d|n} \varphi(d) = n.$$

*Dimostrazione.* Si definisca la funzione

$$f(n) = \sum_{d|n} \varphi(n)$$

e si dimostri innanzitutto che $f(mn) = f(m)f(n)$ per ogni coppia $(m, n)$ di interi positivi coprimi: sia $d$ un divisore proprio di $mn$ e $d = d_1 \cdot d_2$ con $d_1|m$ e $d_2|n$. Da $MCD(d_1, d_2) = 1$ segue che

$$\varphi(d) = \varphi(d_1) \cdot \varphi(d_2)$$

e quindi

$$f(mn) = \sum_{d_1|m} \sum_{d_2|n} \varphi(d_1)\varphi(d_2) = \sum_{d_1|m} \varphi(d_1) \cdot \sum_{d_2|n} \varphi(d_2) = f(m) \cdot f(n).$$

Sia ora $n = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$ la decomposizione in fattori primi di $n$; i divisori di $p_i^{\alpha_i}$ sono tutti e soli i $p_i^j$ per $0 \leq j \leq \alpha_i$; da ciò segue che:

$$f(p_i^{\alpha_i}) = \sum_{j=0}^{\alpha_i} \varphi(p_i^j) = 1 + \sum_{j=1}^{\alpha_i} (p_i^j - p_i^{j-1}) = p_i^{\alpha_i}$$

e quindi

$$f(n) = \prod f(p_i^{\alpha_i}) = \prod p_i^{\alpha_i} = n.$$

□

*Dimostrazione del Teorema 2.2.1.* Sia $a \in F_q^*$ di ordine $d$; si considerino

$$a, a^2, \ldots, a^d,$$

che sono elementi distinti di $F_q^*$ e che, quindi, sono tutte e sole le $d$ radici del polinomio $x^d - 1$; da ciò si deduce che tra di essi ci sono tutti e soli gli elementi di $F_q^*$ di ordine $d$.

$a^j$ ha ordine $d$ se e solo se $MCD(j, d) = 1$, infatti se $MCD(j, d) = d' > 1$, $a^j$ ha ordine $d/d' < d$; se invece $MCD(j, d) = 1$ e se, per assurdo, $a^j$ avesse ordine $\bar{d} < d$, esisterebbero $u, v \in \mathbb{Z}$ tali che $uj + vd = 1$ e quindi

$$(a^{\bar{d}})^{ju} = 1 = (a^{\bar{d}})^{-vd} \Longleftrightarrow (a^{\bar{d}})^{ju+vd} = 1 \Longleftrightarrow$$
$$a^{\bar{d}} = 1$$

assurdo perché $\bar{d} < d$.

Di conseguenza, per ogni divisore $d$ di $q - 1$ si hanno due possibilità: o nessun elemento di $F_q^*$ ha ordine $d$ o $\varphi(d)$ elementi di $F_q^*$ hanno ordine $d$. Dal Lemma, si ha

$$\sum_{d|q-1} \varphi(d) = q - 1,$$

che è l'ordine di $F_q^*$, e quindi per ogni divisore $d$ di $q - 1$ ci devono essere $\varphi(d)$ elementi di ordine $d$. □

**Definizione 2.2.1.** Si definisce *indice di* $a \in F_q^*$ l'ordine del campo quoziente $F_q^* / < a >$. Si potrebbe anche dire che l'indice di $a$ è pari a

$$\frac{q-1}{|a|}.$$

*Osservazione.* Se un intero $l$ divide l'indice di $a \in F_q^*$, allora $l|q-1$ e

$$a^{(p-1)/l} = 1_{F_q}.$$

## 2.3 Congruenze

In questo paragrafo richiameremo alcuni concetti sugli anelli delle classi resto modulo $n$ e vedremo un algoritmo di moltiplicazione in $\mathbb{Z}_m$.

**Definizione 2.3.1.** Si denota con $\mathbb{Z}_n$ l'anello commutativo dei residui modulo $n$; dato un intero $a$, si indica con $b = a \pmod{n}$ quell'unico rappresentante $b$ della classe di equivalenza di $a$ minore di $n$.

**Definizione 2.3.2.** Siano $a, n \in \mathbb{N}$; se $\exists b \in \mathbb{N}$ tale che

$$a \equiv b^2 \pmod{n},$$

si dice che $a$ è un *residuo quadratico* modulo $n$.

*Osservazione.* Una domanda pertinente è : quanti sono i residui quadratici nel campo $\mathbb{Z}_p$ per $p \neq 2$?

Osserviamo che $b^2 = (-b)^2$, pertanto si possono considerare come quadrati di $F_p$ solo i $b^2$ per $b = 1, \ldots, (p-1)/2$. Questo dice chiaramente che i residui in $F_p$, per $p \neq 2$, sono $(p-1)/2$ e i non residui sono $(p-1)/2$.

Richiamiamo il Piccolo Teorema di Fermat, il Simbolo di Legendre e il simbolo di Jacobi, che verranno utilizzati nell'ambito dei test di primalità (paragrafo 6.2).

**Teorema 2.3.1 (Piccolo teorema di Fermat).** *Sia $p \in \mathbb{N}$ un primo; allora ogni intero $a$ soddisfa la congruenza:*

$$a^p \equiv a \pmod{p}$$

*ed ogni intero $a$ non divisibile per $p$ soddisfa la:*

$$a^{p-1} \equiv 1 \pmod{p}$$

**Definizione 2.3.3 (Simbolo di Legendre).** : $\forall a, p \in \mathbb{N}$, con $p$ primo diverso da 2, si definisce il *simbolo di Legendre*:

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{se } p|a; \\ 1, & \text{se } a \text{ è un residuo quadratico} \pmod{p}; \\ -1, & \text{altrimenti.} \end{cases}$$

**Definizione 2.3.4 (Simbolo di Jacobi).** Siano $a, n \in \mathbb{N}$, con $n$ dispari, la cui fattorizzazione in primi sia $n = p_1^{\alpha_1}, \ldots, p_r^{\alpha_r}$. Si definisce il *simbolo di Jacobi* $\left(\frac{a}{n}\right)$ il:

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \cdots \left(\frac{a}{p_r}\right)^{\alpha_r}$$

dove $\left(\frac{a}{p_i}\right)$ è il simbolo di Legendre.

*N.b.* Se $\left(\frac{a}{n}\right) = 1$ per $n$ composto, non è detto che $a$ sia un residuo quadratico modulo $n$, infatti

$$\left(\frac{2}{15}\right) = \left(\frac{2}{3}\right)\left(\frac{2}{5}\right) = (-1)(-1) = 1$$

e non esiste alcun $x \in \mathbb{N}$ tale che $x^2 \equiv 2 \pmod{15}$.

Il metodo dei quadrati ripetuti per l'esponenziazione in $\mathbb{Z}_m$ è un metodo più veloce per calcolare $b^n \pmod{m}$ rispetto alla moltiplicazione ripetuta:

**Teorema 2.3.2.** *Dati $n, m \in \mathbb{N}$ e $b \in \mathbb{Z}$, si ha:*

$$\textsc{time}(b^n \pmod{m}) = O((\log n)(\log^2 m)).$$

*Dimostrazione.* Si supponga che $b < m$ e che ad ogni moltiplicazione si riduca il risultato parziale modulo $m$. Sia $a$ tale risultato parziale.

Sia $n = \sum 2^i n_i$ per $n_j = 0, 1$, di modo tale che possa essere riscritto in notazione binaria come $(n_0, \ldots, n_k)$.

*a.* Sia $a = 1$, $j = 0$, $b_0 = b$;

*b.* Se $n_j = 1$ allora si sostituisca $a$ con $ab_j \pmod{m}$, se no $a$ rimanga invariata;

*c.* Si calcoli $b_{j+1} = b_j^2 \pmod{m}$;

*d.* Si sostituisca $j$ con $j + 1$ e si torni al punto b.

Si osservi che al passo $(j-1)$ si sta calcolando

$$b_j \equiv b^{2^j} \pmod{m}.$$

Se $n_j = 1$, cioè se $2^j$ si trova nell'espansione binaria di n, allora si include $b_j$ nel prodotto per $a$. Dopo $k-1$ passi si ha

$$a = b^n \pmod{m}.$$

□

*N.b.* Il valore di TIME($b^n \pmod{m}$) differisce spesso da un testo all'altro, ma l'algoritmo proposto è sempre lo stesso: un metodo è quello di considerare $m$ costante, da cui $O((\log n)(\log^2 m)) = O((\log n))$; un altro metodo [Kn69] ha come risultato un numero massimo di moltiplicazioni modulo $p$, pari a $2[\log_2 p + 1]$ con un'occupazione di spazio di $3[\log_2 p] + 1$ bit, dove $[x]$ è la parte intera di $x$.

## 2.4 Curve ellittiche

**Definizione 2.4.1.** Dato un campo $K$, una cubica $f(x,y) \in K[x,y]$ e l'insieme

$$C = \{(x,y) \in K \times K | f(x,y) = 0\},$$

si dice che un punto $P \in C$ è un *punto singolare* se ogni retta per $P$ interseca $C$ in un solo altro punto.

**Teorema 2.4.1.** *Non è difficile dimostrare che un punto $(x_0, y_0)$ della cubica $C$ è non singolare se e solo se non è una radice multipla di $f(x,y)$ se e solo se*

$$\frac{\partial f}{\partial x}(x_0, y_0) \neq 0 \quad \textit{oppure} \quad \frac{\partial f}{\partial y}(x_0, y_0) \neq 0.$$

**Definizione 2.4.2.** Sia K un campo; dato un insieme ordinato

$$(a_1, \ldots, a_6) \in K^6,$$

tale che la cubica

$$y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6$$

non abbia punti singolari, si definisce *curva ellittica* o *insieme dei punti della curva ellittica* l'insieme

$$E = \{(x, y) \in K \times K | y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\infty\}.$$

L'elemento $\infty$ è detto *punto all'infinito*. Gli indici indicano il "peso" degli $a_i$ se riguardati come indeterminate; i pesi sono scelti in modo che l'equazione sia omogenea per $x$ di peso 2 e $y$ di peso 3.

L'equazione che definisce la curva ellittica può essere ridotta attraverso una serie di cambi di coordinate, ottenendo così:

$$E = \begin{cases} \{(x, y) \in K^2 \mid y^2 = x^3 + ax + b\} \cup \{\infty\} & \text{char } K \neq 2, 3; \\ \{(x, y) \in K^2 \mid y^2 + y = x^3 + ax + b\} \cup \{\infty\} = & \\ \{(x, y) \in K^2 \mid y^2 + cxy + dy = x^3 + ax + b\} \cup \{\infty\} & \text{char } K = 2; \\ \{(x, y) \in K^2 \mid y^2 = x^3 + cx^2 + ax + b\} \cup \{\infty\} & \text{char } K = 3, \end{cases}$$

Le 3 equazioni in $K[x]$ sopra presentate sono dette *equazioni di Weierstrass*.

*Osservazione.* Per char $K \neq 3$, la richiesta che $E$ non abbia punti singolari è equivalente alla $4a^2 + 27b^3 \neq 0$, per il Teorema 2.4.5.

### 2.4.1 Il gruppo delle curve ellittiche

Nel paragrafo precedente abbiamo introdotto l'insieme delle curve ellittiche $E$; ora introduciamo un'operazione interna, detta somma della corda e della tangente, grazie alla quale $(E, +)$ è un gruppo abeliano.

In ambito crittografico le curve ellittiche sono molto importanti, sia perché esse formano una famiglia di gruppi abeliani molto vasta, sia perché sono molto simili ai gruppi moltiplicativi dei campi finiti $F_q$ sui quali sono definite e quindi facili da utilizzare, ma, nello stesso tempo, sono anche diverse dai normali gruppi $F_q^*$, e pertanto non soddisfano alcune proprietà che, come vedremo, renderebbero troppo facile il lavoro di chi cerca di rompere i sistemi crittografici.

Fino ad oggi non è stato trovato un buon sistema per calcolare l'ordine di questi gruppi abeliani in modo sistematico e conveniente (in termini di tempo): presenteremo i risultati più interessanti fino ad oggi raggiunti, le cui dimostrazioni si trovano in [K87] e [Hu87].

**Definizione 2.4.3.** Dato un punto $P = (x, y) \in E - \{\infty\}$ si definisce il suo *opposto*

$$-(x, y) = (x, -y).$$

E' ovvio dalle equazioni di Weierstrass che

$$(x, -y) \in E \quad \text{se e solo se} \quad (x, y) \in E.$$

Per $P = \infty$, si definisce $-P = \infty$.

**Definizione 2.4.4.** Si supponga di avere due punti $P_1$, $P_2 \in E - \{\infty\}$. Si ha una semplice *somma*: si definisce il punto $P_3 = P_1 + P_2$ come il negativo del terzo punto dell'intersezione tra $E$ e la retta definita da $P_1$ e $P_2$. Questa è detta "legge della corda e della tangente".

*Osservazione.* *a.* Se la retta definita da $P_1$ e $P_2$ è tangente ad uno dei due punti, questo sarà il risultato della somma.

*b.* Se $P_1 = P_2$ la retta da considerare sarà la retta tangente a $P_1$.

*c.* Se $P_1 = -P_2$ allora si pone $P_3 = \infty$. Il punto all'infinito è quindi l'elemento neutro di questa operazione e per questo viene indicato anche con $O$.

Siano ora $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ e $P_1 + P_2 = (x_3, y_3)$, punti di una curva ellittica definita su un campo di caratteristica diversa da 3 (non analizziamo questo caso perché poco interessante sia dal punto di vista generale, sia dal punto di vista computazionale). Vogliamo presentare $x_3$ e $y_3$ in funzione di $x_1, x_2, y_1$ e $y_2$. Si supponga innanzitutto che $P_1 \neq -P_2$ e $P_1, P_2 \neq \infty$.

Sia quindi $y = \alpha x + \beta$ l'equazione della retta $\ell$ passante per i due punti $P_1$ e $P_2$, con $\alpha = (y_2 - y_1)/(x_2 - x_1)$ e $\beta = y_1 - \alpha x_1$. A questo punto $P_3$ ha coordinate $(x, \alpha x + \beta)$ che risolvono l'equazione di Weierstrass insieme a $P_1$ e $P_2$.

Si ricava facilmente:

$$\begin{aligned} x_3 &= \gamma^2 + c\gamma - x_1 - x_2 \\ y_3 &= -y_1 + \gamma(x_1 - x_3) - cx_3 - d. \end{aligned}$$

dove:

$$\gamma = \begin{cases} \left(\frac{y_2 - y_1}{x_2 - x_1}\right) & \text{per } P_1 \neq P_2 \\ \left(\frac{3x_1^2 + a - cy_1}{2y_1 + cx_1 + d}\right) & \text{per } P_1 = P_2. \end{cases}$$

Si dimostra il seguente teorema:

**Teorema 2.4.2.** *L'insieme dei punti di una curva ellittica costituisce un gruppo abeliano con l'operazione definita dalla somma della corda e della tangente.*

Usando queste formule, con il metodo delle quadrature ripetute (teorema 2.3.2), per ogni punto $P \in E$ e per ogni intero $m$ è possibile calcolare $mP$ in $O(\log m)$ operazioni binarie.

Data una curva ellittica $E$ definita su $F_q$, si chiede se è possibile calcolare $N = |E|$. Si osserva che l'elemento $\infty \in E$ e che per ogni $(x, y)$ che soddisfa l'equazione di Weierstrass, anche $(x, -y)$ soddisfa la medesima equazione. Poiché ogni $y$ determina univocamente un $x$ (se esiste), in $E$ ci sono al massimo $2q + 1$ elementi. Ma solo la metà degli elementi di $F_q^*$ hanno radici quadrate, quindi il numero di coppie cercate dovrebbe essere circa la metà di $2q + 1$. Si dimostra che:

**Teorema 2.4.3 (di Hasse).** *Sia $E$ una curva ellittica sul campo finito $F_q$ e $N = |E| \Rightarrow$*

$$N = |E| = q + 1 - t, \ \textit{con} \ |t| \leq 2\sqrt{q}.$$

*Osservazione.* Il risultato di Hasse ci dice che una curva ellittica $E$ definita su un campo finito $F_q$ ha asintoticamente ordine $q - 1$, uguale a quello del gruppo moltiplicativo $F_q^*$.

Se si ha una curva ellittica definita su un campo finito $F_q$, allora è anche definita su $F_{q^r}$. Se si indica con $N$ l'ordine di $E$ su $F_q$ e con $N_r$ l'ordine di $E$ su $F_{q^r}$, un teorema di Weil-Deligne dà un algoritmo molto semplice per calcolare $N_r$ da $N$.

*Osservazione.* René Schoof, nel 1985, ha presentato un algoritmo che permette di calcolare $N$ in $O(\log^9 q)$ operazioni binarie. Nella pratica l'algoritmo è di difficile applicazione.

Un'altra questione importante è la struttura del gruppo abeliano $E$:

**Teorema 2.4.4 (di Cassels).** *Il gruppo $E(F_p)$ è o ciclico o prodotto di due gruppi ciclici di ordine $m_1$ e $m_2$, dove $m_1$ divide sia $m_2$ che $MCD(|E|, p-1)$.*

*Osservazione.* Normalmente si ha $m_1 << m_2$, per cui, $E$ è o un gruppo ciclico o un gruppo "quasi ciclico".

### 2.4.2 Proprietà delle curve ellittiche

In questo paragrafo presenteremo le curve ellittiche supersingolari, che sono molto studiate in ambito crittografico, soprattutto perché il loro grup-

po è facilmente gestibile. Per questo abbiamo bisogno di alcune definizioni preliminari riguardanti le cubiche:

**Definizione 2.4.5.** Si consideri l'equazione cubica:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6;$$

si definiscono i coefficienti $b_i$ e $c_i$:

$$\begin{aligned} b_2 &= a_1^2 + 4a_2 & b_4 &= a_1a_3 + 2a_4 \\ b_6 &= a_3^2 + 4a_6 & b_8 &= a_1^2a_6 - a_1a_3a_4 + 4a_2a_6 + a_2a_3^2 - a_4^2 \\ c_4 &= b_2^2 - 24b_4 & c_6 &= -b_2 + 36b_2b_4 - 216b_6 \end{aligned}$$

dai quali è possibile definire il *discriminante della cubica*

$$\Delta = -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6 = \frac{c_4^3 - c_6^2}{12^3}$$

e la *j-invariante*

$$j(E) = j = \frac{c_4^3}{\Delta} = 12^3\frac{c_4^3}{c_4^3 - c_6^2}.$$

Se la cubica definisce una cubica $E$, allora si parla di *discriminante della curva ellittica* $E$ e la $j$-invariante è invariante per isomorfismi di curve ellittiche.

*Osservazione.* Se una curva ellittica $E$ è definita su un campo $K$ di caratteristica diversa da 2 e da 3, essa ha equazione $y^2 = x^3 + a_4x + a_6$, da cui $c_4 = -48a_4$, $c_6 = -864a_6$ e, quindi,

$$\Delta = -16(4a_4^3 + 27a_6^2).$$

Per $\operatorname{char} K = 3$, si ha $\Delta = a_2^2a_4^2 - a_2^3a_6 - a_4^3$.

**Teorema 2.4.5.** *Una cubica è non singolare se e solo se* $\Delta \neq 0$.

La definizione di curva ellittica richiede la non singolarità della cubica che la definisce; in realtà è possibile definire la somma della corda e della tangente anche sugli insiemi delle radici delle cubiche singolari, pertanto alcuni testi le includono nella definizione 2.4.2; ma il teorema seguente dimostra il perché della nostra scelta:

**Teorema 2.4.6 (di Husemoeller).** *Il gruppo dei punti di una curva ellittica singolare è isomorfo o al gruppo additivo o al gruppo moltiplicativo del campo sul quale è definita la curva.*

Esiste un'altra sottofamiglia di curve ellittiche isomorfe a gruppi moltiplicativi di campi finiti:

**Definizione 2.4.6.** Si dice che una curva ellittica $E$ è *supersingolare* se

$$\operatorname{End} E = \{f : E \longrightarrow E \text{ tale che } f \text{ sia un morfismo}\}$$

è un anello non commutativo, dove la somma di due endomorfismi è data dalla somma delle immagini ed il prodotto è la consueta composizione di funzioni.

**Teorema 2.4.7.** *Una curva ellittica $E$ è supersingolare se e solo se*

$$j(E) = 0.$$

**Teorema 2.4.8 (di Menezes, Okamoto e Vanstone).** *Se $E$ è una curva ellittica supersingolare su un campo $F_q$, con $|E| = q+1$, allora esiste un intero $r$ per cui $E$ è isomorfo ad un sottogruppo di $F_r^*$.*

### 2.4.3 La scelta della curva e di un suo punto

Questo problema non è di facile soluzione, soprattutto se si cerca una curva ellittica che soddisfi alcune proprietà particolari. Ci sono fondamentalmente due metodi per scegliere una curva ellittica $E$ ed un suo punto $P$.

Il primo metodo consiste nello scegliere un campo $F_q$, scegliere tre suoi elementi $x, y, a$ (nel caso $\operatorname{char} K = 3$ si sceglie anche $c$), risolvere l'equazione di Weierstrass in $b$ e verificare che siano soddisfatte le proprietà richieste dalla definizione 2.4.2; in caso contrario si riparte dall'inizio con una scelta diversa delle $x, y, a$. Con questa costruzione si determina un punto della curva: $P = (x, y)$.

Un altro metodo per scegliere $E$ e $P$ è la costruzione di $E \pmod p$: si costruisca $E$ su un campo di numeri (si costruisce così una curva ellittica "globale", normalmente si utilizza il campo $\mathbb{Q}$), e sia $P$ un punto di ordine infinito su $E$.

**Esempio 2.4.1.** Si dimostra che il punto $P = (0, 0)$ sulle curve

$$E : y^2 + y = x^3 - x \quad \text{e} \quad E : y^2 + y = x^3 + x^2$$

è un punto di ordine infinito e genera l'intero gruppo dei numeri razionali.

Ora si scelga un primo $p$ e si consideri la riduzione di $E$ e di $P$ modulo $p$, ossia si riducano i coefficienti dell'equazione di $E$ modulo $p$ e, attraverso un cambio di variabili, si porti l'equazione alla forma della definizione 2.4.2; nella maggioranza dei casi questa curva è non singolare [K87].

La curva ottenuta da questa equazione viene denotata con $E \pmod p$ sul campo $F_p$. Se si riducono le coordinate di $P$ modulo $p$ si ottiene $P \pmod p$, punto di $E \pmod p$.

# 3

# Teoria dei codici

La Teoria dei Codici rappresenta una parte della Teoria della Comunicazione proposta da C.E. Shannon a partire dal 1948, che è una formulazione matematica di concetti oggi assai importanti.

Toccheremo poche nozioni di Teoria della Comunicazione, per soffermarci sui codici correggibili, con i quali sarà possibile, come vedremo nel paragrafo 5.4, costruire un sistema crittografico.

## 3.1 Definizioni

Definiamo innanzitutto il concetto di *messaggio*:

**Definizione 3.1.1.** Dato un alfabeto finito $\Sigma$, si definisce *sorgente* $\mathcal{S}$ un flusso di lettere di $\Sigma$.

Sia $X_k$ il simbolo $k$-esimo prodotto dalla sorgente; si definisce $p_j$ come la probabilità che $X_k = \sigma_j$, per $\sigma_j \in \Sigma$. Se $p_j$ è indipendente da $k$ e dalle $X_i$ per $i \neq k$, allora $\mathcal{S}$ è detta *sorgente a memoria zero*.

Ogni sequenza di lettere di $\Sigma$ prodotte dalla sorgente è detta *messaggio*.

Il messaggio può essere scambiato tra due persone, attraverso un *canale di comunicazione*:

**Definizione 3.1.2.** Dati due alfabeti $\Sigma_1$ e $\Sigma_2$, una qualunque "scatola" che accetti come input parole di $\Sigma_1^*$ e che restituisca come output parole di $\Sigma_2^*$, in modo tale che ad ogni lettera di input corrisponda una lettera di output, è detta *canale di comunicazione*.

Siano $\Sigma_1 = \{a_1, \dots, a_m\}$ e $\Sigma_2 = \{b_1, \dots, b_n\}$; sia $u_1 \cdots u_N$ un generico input di lunghezza $N$, e sia $v_1 \cdots v_N$ il suo corrispondente output; si definisce

$$p_{i,j} = P(v_k = b_j | u_k = a_i)$$

la probabilità che, per $a_i$ input $k$-esimo, $b_j$ sia l'output $k$-esimo. La matrice $P = [p_{i,j}]$ tale che $\sum_j p_{i,j} = 1$ è detta *matrice stocastica*.

Se $p_{i,j}$ è indipendente da $k$, allora il canale di comunicazione è detto *a memoria zero* e i suoi simboli di output dipendono esclusivamente dall'input ricevuto.

Se la sequenza $u_1 \cdots u_N$ corrisponde ad un messaggio, allora si dice che il messaggio viene *trasmesso*.

La matrice stocastica può essere vista come l'elenco delle probabilità che una certa lettera $a_i$ venga trasmessa in modo errato o corretto, come illustra l'esempio seguente:

**Esempio 3.1.1 (Canale simmetrico binario).** Siano $\Sigma_1 = \Sigma_2 = \{0, 1\}$ e sia

$$P = \begin{bmatrix} q & p \\ p & q \end{bmatrix};$$

si stabilisce che la "trasmissione corretta" fa corrispondere l'output 0 all'input 0 e l'output 1 all'input 1, e quindi $p$ è la probabilità dell'errore di trasmissione. Se $P = I$, matrice identità, allora si dice che *il canale non è rumoroso*.

**Definizione 3.1.3.** Dato un canale a memoria zero con alfabeti di input ed output $\Sigma_1$ e $\Sigma_2$, si dice *codice di lunghezza n* ogni collezione $\mathcal{C}$ di elementi di $\Sigma_1^n$ detti *parole codice*. Questo tipo di codice è anche detto *codice a blocchi*.

Per $\Sigma_1 = \Sigma_2 = \Sigma$ e $|\Sigma| = q$; ogni codice di lunghezza $n$ è anche detto *codice q-ario di lunghezza n*.

Se $\mathcal{C} = \{c_1, \ldots, c_N\}$, si dice *regola di decodifica* ogni partizione di $N$ sottoinsiemi di $\Sigma_2^*$

$$\Sigma_2^* = \bigcup_{j=1}^{N} R_j,$$

dove la parola ricevuta $y \in R_k$ viene *decodificata* come $c_k$.

La scelta della regola di decodifica è cruciale per il buon funzionamento di ogni sistema di comunicazione.

*Osservazione.* Si osservi che se vengono dati due alfabeti $\Sigma_1$ e $\Sigma_2$ di uguale ordine $q$, i loro elementi risultano essere in corrispondenza biunivoca e quindi è possibile indicare l'insieme delle parole di lunghezza $n$ su un alfabeto di ordine $q$ come $V_n(q)$. Per $q$ primo si può vedere $V_n(q)$ come spazio vettoriale di dimensione $n$ su $F_q$.

Un messaggio può essere trasmesso in maniera "scorretta", di conseguenza risulta utile stabilire la *distanza* tra due messaggi (ossia tra quello corretto e quello ricevuto). Per questo motivo sono state proposte diverse definizioni; a titolo di esempio, ne presentiamo soltanto una che si basa su alfabeti binari:

**Definizione 3.1.4.** Siano $x, y \in V_n(2)$, con:

$$x = (x_1, \dots, x_n) \quad \text{e} \quad y = (y_1, \dots, y_n);$$

il numero di elementi per i quali $x$ ed $y$ differiscono è detto *distanza di Hamming*:

$$d(x, y) = |\{(x_i, y_i | x_i \neq y_i, \text{ per } i = 1, \dots, n\}|$$

Una regola naturale di decodifica è la *regola della minima distanza*: si decodifica ogni $y$ ricevuta in una arbitraria parola codice $\bar{c}$ che soddisfi la:

$$d(y, \bar{c}) = \min_{c \in \mathcal{C}} d(y, c). \tag{3.3.1}$$

Un codice può pertanto essere corretto se ad ogni $y$ ricevuto si fa corrispondere un'unica parola codice.

## 3.2 Codici correggibili e codici lineari

Diamo ora alcuni metodi per correggere i codici e definiremo una particolare famiglia di codici dove questi metodi non richiedono troppo spazio.

**Definizione 3.2.1.** Sia $\mathcal{C}$ un codice; si definisce la *distanza minima di* $\mathcal{C}$ la

$$d(\mathcal{C}) = \min_{c_i, c_j \in \mathcal{C}} d(c_i, c_j)$$

Quando si riceve un vettore, che dovrebbe rappresentare una parola codice, è possibile correggerlo prendendo al suo posto la parola codice più vicina. Il teorema seguente prova che dei codici facilmente correggibili hanno distanza minime grandi.

**Teorema 3.2.1.** *Sia $\mathcal{C}$ un codice con distanza minima $d$; lo schema di decodifica a distanza minima correggerà fino a $(d-1)/2$ errori.*

*Dimostrazione.* Sia $e$ la parte intera di $[(d-1)/2]$, e sia, per $x \in \mathcal{C}$,

$$S(x, e) = \{y | d(x, y) \leq e\}$$

un intorno chiuso di $x$. Per ipotesi si ha

$$S(x,e) \cap S(z,e) = \emptyset \qquad \text{per } x, z \in \mathcal{C} \text{ e } \ x \neq z$$

e quindi la decodifica a distanza minima correggerà fino ad $e$ errori. □

**Definizione 3.2.2.** Un codice $\mathcal{C}$, con $M$ parole codice di lunghezza $n$ e distanza minima $d$ è detto $(n, M, d)$-codice.

Si dice *codice perfetto* ogni codice $\mathcal{C}$ definito su $\Sigma^n$ tale che esiste un $t > 0$ con le seguenti proprietà:

$$S(x,e) \cap S(z,e) = \emptyset \qquad \text{per } x \neq z$$
$$\bigcup_{x \in \mathcal{C}} S(x,t) = \Sigma^n$$

Questo codice può correggere fino a $t$ errori.

**Esempio 3.2.1.** Un esempio banale di codice perfetto è quello costituito da una sola lettera.

Il metodo più semplice per utilizzare i codici correggibili è quello di costruire una tabella con tutte le possibili parole codice di modo da comparare la con il messaggio ricevuto. Salvo casi particolari, questo metodo richiede troppo spazio per poter essere conveniente; per questo si studiano codici strutturati come i codici lineari.

**Definizione 3.2.3.** Se $\Sigma$ è un alfabeto di cardinalità $q$, le sue lettere sono in corrispondenza biunivoca con gli elementi del gruppo moltiplicativo del campo finito $F_q$, e l'insieme delle parole di lunghezza $n$, $\Sigma^n$, è in corrispondenza biunivoca con lo spazio vettoriale su $F_q$ di dimensione $n$

$$V_n(q) = \{(x_1, \ldots, x_n | x_i \in F_q\}.$$

Si definisce *codice lineare* ogni sottospazio vettoriale $\mathcal{C}$ di $V_n(q)$; se $\mathcal{C}$ ha dimensione $k$, allora è detto $[n,k]$-*codice lineare*, o, se $d$ è la distanza minima, $[n,k,d]$-*codice*.

Un $[n,k,d]$-codice lineare su $F_q$ ha $q^k$ elementi, e pertanto è un $(n, q^k, d)$-codice lineare.

Per il teorema seguente è utile la seguente definizione:

**Definizione 3.2.4.** Dato un spazio vettoriale $V$, si dice *peso* di un suo elemento $x$ il numero delle coordinate non nulle e lo si denota con $w(x)$.

**Teorema 3.2.2.** *La distanza minima di un codice lineare $\mathcal{C}$ è il peso minimo dei suoi vettori non nulli.*

*Dimostrazione.* Sia $d$ la distanza minima di $\mathcal{C}$, e siano $x$ ed $y$ due parole codice tali che $d(x, y) = d$. Dal momento in cui $\mathcal{C}$ è un sottospazio vettoriale, anche $x - y$ è una parola codice e $w(x - y) = d$. Se esistesse un vettore non nullo $z \in \mathcal{C}$ tale che $w(z) < d$, si avrebbe

$$w(z) = d(z, 0) < d \qquad \text{assurdo.}$$

□

**Definizione 3.2.5.** Dati due $(n, M, d)$-codici $\mathcal{C}$ e $\mathcal{C}'$, si dice che sono *equivalenti* se esiste una funzione biunivoca

$$\mathcal{C} \longrightarrow \mathcal{C}'$$

che trasformi ogni parola codice $c$ in un'altra parola codice $c'$ e che sia composizione delle seguenti permutazioni $\pi$:

**permutazione posizionale:** è associata a permutazioni $\pi \in S_n$; si indichino con $c = (c_1, \ldots, c_n)$ e $c' = (c'_1, \ldots, c'_n)$ le parole codice di $\mathcal{C}$ e $\mathcal{C}'$ e si ha $c' = \pi(c)$ se

$$c'_i = c_{\pi(i)}$$

per ogni $c \in \mathcal{C}$;

**permutazione simbolica:** è associata a permutazioni $\pi \in S_M$; si indichino con $c = (c_1, \ldots, c_n)$ e $c' = (c'_1, \ldots, c'_n)$ le parole codice di $\mathcal{C}$ e $\mathcal{C}'$ e si ha $c' = \pi(c)$ se per un certo $j$, con $1 \leq j \leq n$ si ha

$$c'_i = c_i \quad \text{per} j \neq i \quad \text{e} \quad c'_j = \pi(c_j).$$

**Definizione 3.2.6.** Sia $\mathcal{C}$ un $[n, k]$-codice lineare; si definisce essere una *matrice di generatori* una matrice $G$ di dimensioni $k \times n$ le cui $k$ righe rappresentano le coordinate di una base di $\mathcal{C} \subseteq V_n(q)$ rispetto ad una base di $V_n(q)$.

Se $\mathcal{C}$ è equivalente al codice $\mathcal{C}'$, allora si dice che le due matrici di generatori $G$ e $G'$ sono equivalenti.

**Teorema 3.2.3.** *Dato un codice $\mathcal{C}$ con matrice di generatori $G$, la matrice $G'$ ottenuta da $G$ attraverso la combinazione di*

a. *permutazioni tra righe o colonne,*

b. *moltiplicazioni degli elementi di una colonna o di una riga per uno scalare non nullo*

c. *somme di una riga per il multiplo scalare di un'altra riga*

*è matrice di generatori di un codice $\mathcal{C}'$ equivalente a $\mathcal{C}$.*

Si verifica facilmente che un'opportuna combinazione delle operazioniv sopra descritte riduce qualsiasi matrice di generatori ad una matrice di tipo $[I_k|A]$, dove $I_k$ è la matrice identica di tipo $k \times k$:

**Teorema 3.2.4.** *Ogni $[n,k]$-codice lineare $\mathcal{C}$ può essere visto nella sua* forma canonica*, ossia con matrice di generatori di tipo $[I_k|A]$.*

Ora è possibile vedere un utilizzo dei codici lineari: sia $m$ un messaggio di lunghezza $k$ (o ne consideriamo un blocco di lunghezza $k$), $m = (m_1, \dots, m_k)$; sia $\mathcal{C}$ un $[n,k]$-codice lineare su $F_q$ con matrice di generatori definita da:

$$G = \begin{bmatrix} r_1 \\ \cdot \\ \cdot \\ \cdot \\ r_k \end{bmatrix} = [I_k|A],$$

dove $r_i$ è un vettore di lunghezza $n$ e $A$ una matrice di tipo $k \times (n-k)$.

Si ottiene la parola codice

$$c = \sum_{j=1}^{k} m_j r_j,$$

ottenendo quindi un vettore di lunghezza $n \geq k$.

Questa operazione è equivalente alla regola:

$$\begin{aligned} c_i &= m_i \qquad \text{per } (1 \leq i \leq k) \\ (c_{k+1}, \dots, c_n) &= (m_1, \dots, m_k) \cdot A \qquad \text{prodotto di matrici.} \end{aligned}$$

Questo procedimento aumenta la lunghezza del messaggio da trasmettere, aggiungendo una sorta di "coda" (il vettore $(c_{k+1}, \dots, c_n)$) rendendo, però, il sistema più affidabile.

Si osservi ora che:

$$(c_1, \dots, c_k) \cdot A = (m_1, \dots, m_k) \cdot A = (c_{k+1}, \dots, c_n),$$

pertanto

$$[-A^T, I_{n-k}] \cdot c^T = 0,$$

ed è possibile dare la definizione e il teorema seguenti:

**Definizione 3.2.7.** La matrice $H = [-A^T, I_{n-k}]$ è detta *matrice di controllo* del $[n, k]$-codice lineare con matrice di generatori $[I_k, A]$.

**Teorema 3.2.5.** *Sia dato un $[n, k]$-codice lineare $\mathcal{C}$ con matrice di generatori $[I_k, A]$ e matrice di controllo $H$, allora un vettore $z$ è una parola codice di $\mathcal{C}$ se e solo se $Hz^T = 0$.*

**Esempio 3.2.2.** Sia $\mathcal{C}$ un codice su $F_3$ con matrice di generatori

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 2 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 2 & 0 & 1 \end{bmatrix}$$

e sia $m = 102102$, che deve essere diviso in vettori di lunghezza 3: $(1, 0, 2)$ e $(1, 0, 1)$, per poi codificarlo in parole codice:

$$(1, 0, 2) \mapsto r_1 + 2r_3 = (1, 0, 2, 2, 2, 2)$$
$$(1, 0, 1) \mapsto r_1 + r_3 = (1, 0, 1, 0, 2, 1)$$

ottenendo così la sequenza 102 222 101 021. Se si considerano le prime 6 cifre, le prime tre sono le cifre del messaggio, le altre tre sono le cifre per il controllo.

La matrice di controllo di $\mathcal{C}$ è :

$$H = \begin{bmatrix} -1 & 0 & -2 & 1 & 0 & 0 \\ -2 & -1 & 0 & 0 & 1 & 0 \\ 0 & -1 & -1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 1 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 & 1 & 0 \\ 0 & 2 & 2 & 0 & 0 & 1 \end{bmatrix}$$

e le cifre 102 222, che corrispondono alla parola chiave $c$ devono soddisfare l'equazione $Hc^T = 0$, che si riduce a:

$$2c_1 + c_3 + c_4 = 0 \qquad c_1 + 2c_2 + c_5 = 0 \qquad 2c_2 + 2c_3 + c_6 = 0.$$

Se, per esempio, il messaggio ricevuto non soddisfa la prima e la terza equazione, è probabile che ci sia stato un errore nella trasmissione della cifra $c_3$.

### 3.2.1 Decodifica con il metodo della minima distanza per codici lineari

Sia dato un $[n, k]$-codice lineare $\mathcal{C}$ su $\Sigma = F_q$, di modo che $\mathcal{C}$ abbia $q^k$ parole codice di lunghezza $n$ e il numero dei vettori che si possono ricevere sia $q^n$ e supponiamo di lavorare nel caso binario. Possiamo vedere $\mathcal{C}$ come sottogruppo del gruppo abeliano additivo $V_n(q)$ e considerare il gruppo quoziente

$$\frac{V_n(q)}{\mathcal{C}}.$$

Sia $y \in V_n(q)$ il vettore ricevuto. Si dice che $e$ è un *possibile vettore sbagliato di* $y$ se esiste un $c \in \mathcal{C}$ tale che $y - c = e$, quindi $e$ è un vettore sbagliato associato ad un vettore ricevuto se rappresenta un vettore di errori di trasmissione.

*Osservazione.* Se $y$ è un vettore ricevuto, allora l'insieme dei possibili vettori sbagliati è dato dai rappresentanti della classe di equivalenza di $V_n(q)/\mathcal{C}$ con rappresentante $y$. Infatti $e$ è un possibile vettore sbagliato di $y$ se e solo se esiste una parola codice $c$ tale che $e = y - c$ se e solo se $e = y + c'$, dove $c' = -c \in \mathcal{C}$, se e solo se $e \in y + \mathcal{C}$.

Così il problema della decodifica con il metodo della distanza minima si è ridotto alla ricerca del possibile vettore sbagliato di peso minimo $z_0$, che definiamo come rappresentante della classe di equivalenza $y + \mathcal{C}$.

Una volta ricevuto $y$ si cerchi il rappresentante $z_0$ della classe $y + \mathcal{C}$ e si decodifichi $y$ come la parola codice $y - z_0$. Questo metodo ha l'inconveniente che la ricerca di $z_0$ potrebbe richiedere troppo tempo.

**Teorema 3.2.6.** *Due vettori $y_1$ e $y_2$ sono nella stessa classe di equivalenza di $V_n(q)/\mathcal{C}$ se e solo se*

$$Hy_1^T = Hy_2^T$$

*Dimostrazione.* I vettori $y_1$ e $y_2$ sono nella stessa classe di equivalenza di $V_n(q)/\mathcal{C}$ se e solo se esiste una parola codice $c$ tale che

$$y_1 = y_2 + c \text{ e } Hc^T = 0.$$

□

**Definizione 3.2.8.** Si definisce *sindrome della classe di equivalenza* $a + \mathcal{C}$ il vettore $Ha^T$, che è ben definita per il teorema precedente.

Ora si ha una tabella che dà il rappresentante $z_0$ per ogni sindrome e si può accelerare il metodo sopra esposto così:
una volta ricevuto $y$ si calcoli $Hy^T$, dalla tabella si cerchi il corrispondente rappresentante di peso minimo $z_0$ e si decodifichi $y$ come la parola chiave $y - z_0$. Si dimostra che questo è l'algoritmo più veloce per la decodifica con il metodo della distanza minima.

Si noti che il rappresentante con il peso minimo non è unico, ma questo non crea difficoltà. Inoltre si osservi che, nel caso di un $[n, k]$-codice lineare, ci sono $2^{n-k}$ classi di equivalenza di $V_n(q)/\mathcal{C}$ e, pertanto, la ricerca del rappresentante della classe di equivalenza con il peso minimo potrebbe risultare laboriosa per $n$ e $k$ grandi. Malgrado ciò questo metodo è attualmente molto utilizzato nella pratica.

# Parte II

# Crittografia

# 4

# Introduzione alla crittografia

## 4.1 Generalità

La crittografia nasce dalla necessità di avere scambi di informazioni riservate attraverso canali insicuri, quindi dall'assunto che è possibile intercettare un messaggio dal momento in cui viene inviato a quello in cui viene ricevuto, indipendentemente dal fatto che si usi la posta a cavallo o la posta elettronica.

L'intercettatore può agire in due modi: leggere il messaggio senza modificarlo e lasciarlo andare a destinazione (*intercettatore passivo*) o leggere il messaggio e modificarlo, o anche distruggerlo (*intercettatore attivo*).

Un primo obiettivo degli interlocutori "autorizzati" è che l'intercettatore non riesca a capire quanto legge in un tempo tale che l'informazione possa essere utile: per questo è necessario usare qualche metodo di crittazione.

La nozione di "tempo utile" dipende dalla natura del messaggio: un'informazione su un'operazione finanziaria può essere intercettata e decrittata, senza risultare utile, in poche ore; una nuova specialità medica di una casa farmaceutica in mesi; la ricetta della Coca Cola in lustri. Da queste considerazioni dipendono le scelte di chi vuole utilizzare un crittosistema.

La crittografia è sovente associata alla steganografia, ossia l'arte di nascondere messaggi. Classico esempio è l'inchiostro simpatico. Un altro bell'esempio è riportato da Erodoto: Istaio e Aristagora comunicavano in questo modo attraverso i rispettivi Paesi, che erano in guerra tra loro: radevano i capelli di uno schiavo, disegnavano sul cranio alcuni simboli, aspettavano che allo schiavo crescessero i capelli e gli facevano attraversare il confine; al destinatario non restava che rapare di nuovo lo schiavo e leggere il messaggio.

**Definizione 4.1.1.** Il messaggio, nella sua forma originaria, è chiamato *testo*

*in chiaro (plain text)*; il mittente critta il messaggio (vedere la definizione successiva), e ottiene un *testo crittato* o *crittotesto*; il destinatario riceve il messaggio e *decritta il crittotesto*. Mittente e destinatario sono anche detti *utilizzatori legali* o *autorizzati* del sistema crittografico.

Chiunque altro cerchi di ricavare dal testo crittato il testo in chiaro viene detto *crittanalista* e la sua attività *crittanalisi*.

*Osservazione.* Una delle nomenclature che vengono date intuitivamente agli utilizzatori di un crittosistema e ai crittanalisti nasce dalla distinzione tra "buoni" e "cattivi"; in una definizione formale ciò non è possibile, infatti possono inviarsi l'un l'altro messaggi crittati sia gli alleati in una guerra sia i membri di una banda di criminali, possono cercare di decrittare messaggi sia i nemici in guerra sia la polizia.

**Definizione 4.1.2.** Un *sistema crittografico* è la collezione dei seguenti oggetti:

*a.* un insieme $PT$ formato da tutti i possibili testi in chiaro, ossia stringhe $m$ su un prefissato alfabeto $\Sigma$;

*b.* un insieme $CT$ dei crittotesti, ossia stringhe $c$ su un prefissato alfabeto $\Sigma'$;

*c.* un insieme $K$, detto *insieme delle chiavi* e due algoritmi $D$ e $E$:

$$E : K \times PT \longrightarrow CT$$
$$D : K \times CT \longrightarrow PT$$

ogni chiave $k$ in $K$ determina un metodo di crittazione $E(k)$ e un metodo di decrittazione $D(k)$ tale che per ogni elemento $m \in PT$ sia $D(E(m)) = m$ (generalmente, come in questo caso, si sottintende la chiave $k$ e si indicano solo $E$ e $D$).

Coloro che utilizzano un crittosistema devono accordarsi sui dettagli degli oggetti descritti e utilizzarli come convenuto, ossia devono seguire un *protocollo*.

*Osservazione.* In generale, è possibile usare ogni tipo di linguaggio su qualsiasi tipo di alfabeto per gli elementi di $PT$ e di $CT$; risulta però molto più facile per un crittanalista rompere un sistema strutturato sul normale alfabeto a 26 lettere rispetto ad un alfabeto binario, ossia basato su $\Sigma = \{0, 1\}$.

In crittografia si usa la seguente convenzione:

**Definizione 4.1.3.** Quando si trasforma un messaggio usando diversi alfabeti, senza però l'intenzione di renderlo incomprensibile, si dice che lo si *codifica* o *decodifica*.

**Esempio 4.1.1.** Dato un messaggio $m = \sigma_l \dots \sigma_1$ di lunghezza $l$, definito sull'alfabeto inglese A-Z di 26 lettere, si ottiene un numero in base decimale $m'$ trasformando ogni lettera $\sigma_i$ di $m$ nel suo equivalente numerico $\sigma'_i$ (la A corrisponde allo 0 e la Z al 25); la stringa $\sigma'_l \dots \sigma'_1$ può essere vista come un numero in base 26 dal quale si calcola

$$m' = 26^{l-1}\sigma'_l + \dots + 26 \cdot \sigma'_2 + \sigma'_1 \in \{0, \dots, 26^l - 1\}$$

che si può ulteriormente codificare passando dalla base decimale a quella binaria (i numeri da 0 a 25 espressi in base binaria sono stringhe di 5 bit ciascuna).

Diamo infine due definizioni molto importanti per il seguito:

**Definizione 4.1.4.** I sistemi crittografici possono essere suddivisi in due famiglie:

**block cipher:** dato un messaggio $m$ (eventualmente già codificato), lo si spezza in blocchi di lunghezza fissata, che vengono crittati uno ad uno con lo stesso sistema e la stessa chiave $k$. Ogni blocco potrebbe anche essere lungo una lettera;

**stream cipher:** il messaggio $m$ (eventualmente già codificato) viene diviso in blocchi $m_j$ (solitamente di una sola lettera), che vengono crittati con una chiave dipendente da $j$. Questo secondo metodo è oggi ritenuto più efficace del block cipher perché un eventuale errore di trasmissione sarebbe facilmente rilevabile (per le $j$) e quindi non comprometterebbe tutto il lavoro.

Nel seguito, per semplicità, presenteremo esempi di block cipher; per approfondire l'argomento sulle stream cipher suggeriamo il testo [DXS86].

La crittanalisi non persegue l'obiettivo di trovare un algoritmo che, in tempo utile, inverta la funzione usata nel protocollo, posto che si richiede soltanto di ricavare il messaggio in chiaro dal messaggio crittato sfruttando anche le eventuali debolezze del crittosistema.

Presentiamo in proposito una classificazione dei problemi che si presentano al crittanalista:

**Solo crittotesto.** Il crittanalista ha solo il crittotesto; più è complicato il sistema, più lunghi devono essere gli esempi di cui ha bisogno.

**Testo in chiaro conosciuto.** Il crittanalista conosce qualche coppia testo in chiaro - crittotesto $(m, E(m))$.

**Testo in chiaro scelto.** Il crittanalista conosce qualche coppia $(m, E(m))$ con $m$ scelto da lui. Molto utile se ha qualche congettura sulla chiave usata: caso realistico nei sistemi a chiave pubblica.

**Chiave di crittazione.** Il crittanalista conosce $E(k)$ e cerca di calcolare l'algoritmo $D(k)$, normalmente avendo a disposizione esempi di crittotesto. Molto tipico nel caso della chiave pubblica. Il crittanalista ha sovente a disposizione $E(k)$ parecchio tempo prima della sua utilizzazione; in questo caso ha tempo per elaborare alcuni calcoli, ossia di pre-operare.

Si fa uso anche di altre tecniche, tra queste la conoscenza di coppie di bit in chiaro - crittate e di disturbi nel sistema di comunicazione.

Si dice che *il sistema è rotto* se la chiave di crittazione è calcolabile in tempo polinomiale, ovvero ogni crittotesto è riconducibile al messaggio in chiaro avendo a disposizione solo il protocollo e i dati pubblici.

## 4.2 Crittosistemi

Introduciamo innanzitutto i *sistemi crittografici classici* o *a chiave privata*, in cui la chiave $k$ e gli algoritmi $D$ ed $E$ sono noti a tutti gli utilizzatori autorizzati del sistema.

Francis Bacon (Londra 1561 - 1626) propose tre condizioni per avere un buon sistema crittografico classico:

*a.* Gli algoritmi di crittazione e di decrittazione, per chi è autorizzato ad usare il sistema, devono essere facili;

*b.* Senza conoscere la chiave $k$ deve risultare impossibile trovare $m$ da $c$;

*c.* $c$ deve avere un aspetto innocuo (attualmente questa condizione è superflua, visto che si tende ad utilizzare un alfabeto binario nei mezzi di comunicazione informatici).

Il progettista di crittosistemi, oggi, deve quindi tendere a che le prime due condizioni siano soddisfatte; deve inoltre non sottostimare mai il crittanalista (*regola aurea del progettista di crittosistemi*) e deve sempre supporre che questi conosca il crittosistema usato. Infatti la forza di un sistema crittografico deve risiedere nell'impossibilità di trovare sia $D(k)$ sia $m$, e non sulla segretezza di $E(k)$.

Introduciamo ora tre sistemi crittografici classici assai noti.

**Esempio 4.2.1 (Il sistema crittografico Caesar).** Questo sistema è descritto da Svetonio e veniva usato da Giulio Cesare per comunicare con Cicerone: è un semplice esempio di crittografia classica a chiave privata. Utilizzeremo l'alfabeto inglese a 26 lettere (supponiamo di non dovere usare né spazi né punteggiatura); prendiamo come esempio la stringa $m =$ "sistemaCaesar".

Come prima cosa si codifica il testo in chiaro, assegnando ad ogni lettera il suo corrispondente numerico nell'alfabeto $\Sigma = \{0, 1, \ldots, 26\}$, ottenendo la stringa $m' = 19-09-19-20-05-13-01-03-01-05-19-01-18$.

Sia $E(k)$ l'algoritmo di crittazione che ad ogni lettera $\sigma$ sostituisce l'elemento di $\Sigma$ dato da:

$$\sigma' = (\sigma + k) \pmod{26}.$$

Per $k = 3$ (lo stesso valore usato da Cesare...) si ha $c' = 22-12-22-23-08-16-04-06-04-08-22-04-21$; la si può decodificare ottenendo $c =$ "vlvwhpdfdhvdu", che non ha certamente un aspetto innocuo, come voleva Bacon.

Omettendo i calcoli di controllo, è ovvio che l'algoritmo di decrittazione $D(k)$ è :

$$\sigma = (\sigma' - k) \pmod{26}.$$

Evidentemente questo sistema è molto fragile.

**Esempio 4.2.2 (Altri semplici sistemi crittografici).** Nel corso dei secoli sono stati proposti altri sistemi crittografici che rappresentano delle evoluzioni del sistema Caesar: utilizzando le stesse notazioni dell'esempio precedente, si può ottenere un sistema crittografico affine utilizzando un algoritmo $E(a, b)$, dove $MCD(a, 26) = 1$, tale che $\sigma' = (a\sigma + b) \pmod{26}$ oppure si può ottenere un sistema crittografico matriciale scegliendo come chiave $k$ una matrice $n \times n$ e moltiplicandola con blocchi di $n$ lettere di $m$ (visti come matrici $n \times 1$) per $k$ modulo 26.

Altri sistemi crittografici non utilizzano l'algebra, ma diagrammi, mascherine che nascondono una parte del testo, vocabolari segreti, eccetera.

Nel nostro secolo, infine, sono state realizzate macchine per poter rendere sempre più complicati i crittotesti. Una di queste, molto famosa, è la Enigma usata dai tedeschi durante la Seconda Guerra Mondiale. I servizi segreti inglesi, per poter decrittare i messaggi crittati con Enigma, hanno addirittura dovuto inventare il computer.

Questo, poi, ha consentito lo sviluppo di sistemi sempre più complessi per crittare messaggi; uno di essi è il DES.

**Esempio 4.2.3 (Il DES (Data Encription Standard)).** Il DES a 64 bit è un protocollo di tipo block cipher, mediante il quale il messaggio $m$ viene suddiviso in blocchi da 64 bit ciascuno, a loro volta manipolati con permutazioni, somme modulo 2 con funzioni della chiave, ulteriori suddivisioni in sottoblocchi e così via. Questo algoritmo non è stato ancora attaccato efficacemente: il metodo più pratico è quello "a forza bruta", ossia provare tutte le chiavi possibili fino ad ottenere $m$; oggi è possibile decrittare un messaggio $c = E(k)$ con $k$ di 64 bits in 4 giorni, ma per $k$ a 128 bits non è stato raggiunto un risultato accettabile.

Il DES è importante perché rappresenta la prima standardizzazione di sistemi crittografici consentendo così a due utilizzatori qualsiasi di scambiarsi unicamente la chiave del sistema senza doversi scambiare anche l'algoritmo.

La standardizzazione dei sistemi crittografici coinvolge anche la politica, tanto che molti Paesi occidentali ritengono la crittografia un'arma soggetta a regolamentazione: negli Stati Uniti gli studiosi di crittografia vengono schedati e gli algoritmi utilizzabili, insieme alla lunghezza massima delle chiavi, sono definiti per legge, perché i servizi di controspionaggio siano in grado di poter decrittare ogni tipo di crittotesto intercettato. Il DES con chiave a 64 bits è tuttora il sistema crittografico a chiave privata legale negli USA ed è quindi facilmente immaginabile lo scompiglio che ha creato l'annuncio fatto nel luglio del 1998 di poter utilizzare una macchina con un costo inferiore a mezzo miliardo di lire italiane in grado di decrittare in meno di 4 giorni ogni tipo di crittotesto ottenuto da questo sistema.

Ulteriori informazioni sull'algoritmo si ricavano dall'articolo ufficiale del NIST (National Institute of Standards and Technology) [FIPS46]. Questo algoritmo può anche essere utilizzato per altri scopi, dei quali il NIST ne ha classificati quattro [FIPS81], tra cui la generazione di numeri pseudocasuali.

**Esempio 4.2.4 (La base monouso o One-time pad).** Questo sistema è stato introdotto da G. S. Vernam nel 1926 e veniva utilizzato durante la guerra fredda per una delle linee segrete tra Mosca e Washington [SG79].

Sia $\Sigma$ un alfabeto di $n$ lettere, $m = (m_1, \dots, m_N)$ il messaggio in chiaro di lunghezza $N$, $k = (k_1, \dots, k_N)$ una chiave data da una sequenza casuale

di $N$ lettere di $\Sigma$, in modo tale che ognuna di esse abbia probabilità pari a $1/n$ di essere scelta per il posto $i$. Si pone

$$c = E(k, m) = (c_1, \ldots, c_N) \quad \text{dove} \quad c_i = m_i + k_i \pmod{n}.$$

La probabilità che esca una certa chiave $k$ è pari a $|\Sigma|^{-N}$ e i possibili crittotesti sono $|\Sigma|^{-N}$, ognuno ugualmente probabile; in questo caso si dice che questo sistema crittografico è *perfettamente sicuro* [WD88].

Ci si imbatte tuttavia con difficoltà di costruzione e di utilizzo: lo scambio delle chiavi è equivalente allo scambio di messaggi e quindi il sistema risulta non conveniente. Questo problema può essere risolto da due interlocutori che dispongono dello stesso generatore di lettere pseudocasuali (paragrafo 8.3).

Mentre la crittografia classica è basata quasi esclusivamente sull'esigenza della segretezza, i nuovi sistemi crittografici, proposti dalla nascita della crittografia a chiave pubblica, hanno altri utilizzi, tra i quali l'identificazione dell'interlocutore, l'autenticazione e la firma dei messaggi, la creazione di messaggi impossibili da smentire, firme e crittazioni di gruppo.

Conviene dunque distinguere i vari sistemi:

**sistemi a una chiave o simmettrici,** che permettono la crittazione e decrittazione di messaggi con la stessa chiave o con chiavi diverse ma facilmente ricavabili l'una dall'altra; rappresentanti di questi sistemi sono presentati negli esempi precendenti;

**sistemi a due chiavi o asimmetrici,** tra i quali i sistemi a chiave pubblica presentati nel capitolo seguente;

**sistemi a chiave multipla,** usati per i controlli d'accesso, per i controlli condivisi e per le firme di gruppo (capitolo 9);

**sistemi senza chiave,** nei quali, in genere, si utilizzano funzioni impossibili (o quasi) da invertire, come le funzioni hash a senso unico (paragrafo 8.1).

# 5

# Crittografia a chiave pubblica

La crittografia a chiave pubblica trae origine dall'articolo di Diffie e Hellman "New Directions in Cryptography" del 1976 [DH76]. Essa si basa sul fatto che è possibile rendere pubblico un certo tipo di algoritmo a chiave, ma non trovarne l'inverso in tempi utili al crittanalista, grazie all'esistenza di funzioni di cui è computazionalmente troppo lungo calcolare l'inverso.

**Definizione.** Dati due insiemi $X$ e $Y$, si dice *funzione a senso unico* una funzione $f : X \longrightarrow Y$ iniettiva di cui è computazionalmente trattabile il calcolo di $f(x)$ per quasi ogni $x \in X$ e di cui è computazionalmente intrattabile il calcolo di $f^{-1}(y)$ per ogni $y \in f(X)$.

**Esempio 1.** Un semplice esempio è il seguente: fare corrispondere ad ogni lettera il numero di telefono di un abbonato il cui cognome inizi per quella lettera. Senza il servizio 12 risulterebbe assai impegnativo risalire dal numero di telefono al nome dell'abbonato.

**Esempio 2.** Alcune funzioni legate a problemi NP sono funzioni a senso unico: il problema della fattorizzazione e il problema dei pacchetti (knapsack problem, paragrafo 5.3) sono di tipo NP e le funzioni ad essi associate (la moltiplicazione di interi e la somma di elementi distinti di un insieme) sono a senso unico e sono state utilizzate per la costruzione di crittosistemi.

Proviamo ora a dare un'idea di come si costruisce un sistema a chiave pubblica [SA96]:

*a.* Sia $A$ un problema difficile, ossia intrattabile per la teoria della complessità computazionale.

*b.* Sia $A_f$ un caso particolare di $A$ che sia trattabile in tempo polinomiale.

*c.* Sia $A_s$ il problema $A_f$ "scompigliato", di modo tale che $A_s$ non assomigli più ad $A_f$, ma ad un caso generale di $A$ (intrattabile).

*d.* Si pubblicizzi il protocollo con cui si usa $A_s$. Il metodo per passare da $A_s$ a $A_f$ è detto *botola segreta.*

*e.* Si costruiscano i dettagli del crittosistema.

Tuttavia, questa idea è molto astratta, infatti sono stati utilizzati diversi problemi NP per costruire sistemi crittografici come suggerito, ma non tutti hanno funzionato (vedi 5.3).

Ma perché l'introduzione di una crittografia a chiave pubblica? Già nel 1976 Diffie e Hellman davano tre esempi:

*a.* per poter usare un sistema crittografico con molte persone anche distanti geograficamente tra loro, senza dover stabilire incontri preliminari per lo scambio di chiavi;

*b.* per la costruzione di protocolli di identificazione e di firma (capitolo 9);

*c.* per il deposito di password dei computer: se un utente riesce ad accedere alle banche dati di un computer è anche possibile che possa accedere all'elenco delle password legate agli utenti, quindi queste devono essere crittate di modo tale che ogni password $pw$ venga memorizzata come $E(k, pw)$, dove $E$ è un algoritmo di crittazione a chiave (rappresentata da $k$): ogniqualvolta si dà $pw$ al sistema, esso calcola $E(k, pw)$ e la confronta con quanto ha in memoria. Se qualcuno accede alla banca dati delle password può anche accedere all'algoritmo per crittarle; se esso si basa su un sistema crittografico a chiave pubblica, il lavoro per calcolarne l'inverso risulta troppo oneroso.

Con questi sistemi sono anche costruibili protocolli che garantiscano l'autenticità dei dati, o che accertino che il mittente non abbia "barato" inviando dati non completi o non autentici, o che attestino l'integrità del messaggio arrivato. Oggi è utilizzabile un sistema basato sulla chiave pubblica per il commercio elettronico [L97].

Nell'articolo di Diffie e Hellman del 1976 [DH76] sono presentate diverse proposte per lo sviluppo di protocolli crittografici, ma l'unica completamente sviluppata è quella relativa allo scambio di chiavi per sistemi crittografici classici, basata sul problema del logaritmo discreto.

Nel 1978 [RSA78] fu presentato il protocollo crittografico a chiave pubblica oggi più diffuso per scambio di messaggi crittati (e quindi non solo chiavi): l'RSA, basato sulla fattorizzazione di interi positivi.

Solo nel 1984 [ELG84], ElGamal ha presentato un protocollo basato sul logaritmo discreto per lo scambio di messaggi (ma questi due ultimi sistemi sono molto onerosi, perché richiedono un tempo di lavoro almeno 1000 volte superiore al DES).

Un altro crittosistema a chiave pubblica molto noto è quello basato sul "knapsack problem", presentato nel 1978 [MH78], soprattutto perché è stato rotto pochi anni dopo [AS82] con un algoritmo polinomiale, suscitando grande sconforto nella comunità degli studiosi di crittografia che su quel protocollo aveva riposto molta fiducia perché basato su un problema NP (paragrafo 5.3).

Da qui in poi, seguendo la gran parte della letteratura sulla crittografia, chiameremo Alice e Bob due generici utilizzatori di sistemi crittografici.

## 5.1 Logaritmo discreto

> Dato un gruppo finito $G$, dati $y, g \in G$, dire se esiste un $a \in \mathbb{N}$ tale che $g^a = y$ e presentarlo

è detto "problema del logaritmo discreto" e $a$ è il *logaritmo discreto di $y$ in base $g$*.

Se si lavora in un gruppo finito moltiplicativo ($\mathbb{Z}_p^*$ o $F_q^*$, per esempio) è decisamente più veloce calcolare $g^a$ (con il metodo della quadratura ripetuta del paragrafo 2.3.2) rispetto a $\log_g y$, a differenza dell'usuale logaritmo su $\mathbb{R}$, dove i calcoli per $y = g^a$ e per $a = \log_g y$ richiedono tempo analogo.

**Esempio 5.1.1.** Sia $G = F_{19}^* = \mathbb{Z}_{19}^*$, $g = 2$ generatore di $G$, allora $\log_2 7 = 6$.

**Esempio 5.1.2.** In $F_9^*$ sia $\alpha$ radice di $x^2 - x - 1 \in F_9[x]$, allora $\log_\alpha(-1) = 4$.

Infatti, per costruire $F_9$, si scelga un polinomio monico irruducibile di $F_3[x]$, che potrebbe essere $x^2 - x - 1$; le potenze successive della radice $\alpha$ sono quindi (dove $\alpha^2$ è sempre rimpiazzato da $\alpha + 1$, dal momento in cui $x^2 = x + 1$):

$$\begin{aligned} \alpha^2 &= \alpha + 1 & \alpha^3 &= -\alpha + 1 \\ \alpha^4 &= -1 & \alpha^5 &= -\alpha \\ \alpha^6 &= -\alpha - 1 & \alpha^7 &= \alpha - 1 \\ \alpha^8 &= 1, \end{aligned}$$

da dove si deduce che $\alpha$ è generatore di $F_9^*$ e $\log_\alpha(-1) = 4$.

*Osservazione.* Vale la proprietà: $\log_g y = (\log_g h)(\log_h y)$ per ogni $h \in G$.

Per l'applicazione crittografica si richiede che:

*a.* $G$ sia un gruppo finito;

*b.* in $G$ sia facilmente verificabile l'uguaglianza tra elementi;

*c.* in $G$ sia indicata una facile regola per la moltiplicazione.

Inoltre si suppone (per comodità) che il gruppo $G$ sia generato dalla base $g$ e che il gruppo $G$ sia pubblico. Quindi si potrebbe scegliere un gruppo ciclico di cui è difficile calcolare l'ordine, per esempio un sottogruppo di $\mathbb{Z}_n$ con $n = pq$, dove $p$ e $q$ sono due primi (grandi). Per rompere il sistema si dovrebbe fattorizzare $n$ per poter lavorare su $\mathbb{Z}_p^*$ e $\mathbb{Z}_q^*$, generati da $g_p$ e $g_p$: ogni elemento $y$ di $\mathbb{Z}_n$ può essere visto come coppia ordinata $(y_p, y_q) \in \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ e il calcolo dei due logaritmi discreti $\log_{g_p} y_p$ e $\log_{g_q} y_q$ potrebbe essere più veloce del calcolo di $\log_g y$ (con l'algoritmo descritto nel paragrafo 7.2). In questo modo sono stati uniti il problema del logaritmo discreto ed il problema della fattorizzazione.

Si ipotizza che un sistema crittografico basato sul logaritmo discreto risulti facilmente implementabile se costruito su campi $F_{2^n}$ [OD84] e, per rendere questo sistema ancora più veloce per gli utilizzatori legali, sono state presentate diverse soluzioni per l'esponenziazione veloce in tali campi.

Insieme al problema del logaritmo discreto va considerato quello di Diffie-Hellman:

> Dati un gruppo finito $G$, un suo elemento $g$ e due sue potenze $g^a$ e $g^b$ (per $a, b \in \mathbb{Z}$ non noti), trovare $g^{ab}$.

*Osservazione.* Il problema di Diffie - Hellman è riducibile al problema del logaritmo discreto, infatti se si riesce a calcolare $\log_g g^a = a$ e $\log_g g^b = b$ si riesce anche a calcolare $g^{ab}$. Ma è possibile calcolare $g^{ab}$ senza passare dal calcolo del logaritmo discreto di $g^a$ e $g^b$?

De Boer [PC90] ha supposto che se l'intero $\varphi(p-1)$ è liscio (cioé ha fattori primi "piccoli"), allora in $\mathbb{Z}_p^*$ i problemi risultano essere di difficoltà equivalente.

Siamo ora in grado di illustrare i protocolli basati sul logaritmo discreto, mentre nel Capitolo 7 verranno presentati i principali metodi di crittanalisi.

### 5.1.1 Il protocollo per lo scambio di chiavi di Diffie-Hellmann

Sia $F_q$ un campo finito, con $q$ primo di dominio pubblico; sia $g \in F_q$, anch'esso pubblico, generalmente si sceglie $g$ generatore di $F_q^*$.

Alice sceglie un intero positivo $a$, con $a \leq q-1$, calcola $g^a \in F_q$ e lo rende pubblico; Bob sceglie $b \in \mathbb{N}$, con $b \leq q-1$, calcola $g^b \in F_q$ e lo rende pubblico; Alice e Bob useranno la chiave $g^{ab}$ per sistemi crittografici comuni.

Il crittanalista si trova di fronte al problema di Diffie - Hellman.

**Esempio 5.1.3.** Sia $m$ il messaggio da crittare su un alfabeto di 26 lettere $\Sigma$, si supponga di usare un sistema crittografico classico di tipo Caesar (esempio 4.2.1), per cui

$$\sigma' = (\sigma + k) \pmod{26},$$

dove $\sigma'$ e $\sigma$ sono le lettere del testo crittato e del testo in chiaro e $k$ la chiave.

Alice e Bob devono scambiarsi la chiave $k$, quindi scelgono un campo finito ed un suo generatore: $F_{53} = \mathbb{Z}_{53}$ e $g = 2$.

Alice sceglie $a = 29$ e riceve da Bob l'elemento $2^b \pmod{53} = 12$, qunidi calcola $k$:

$$k = 2^{ab} \pmod{53} = 12^a \pmod{53} = 12^{29} \pmod{53} = 21 \in F_{53};$$

analogamente, Bob sceglie $b = 19$, riceve da Alice l'elemento $2^a \pmod{53} = 45$ e calcola $k$ (che è ovviamente uguale a quella calcolata da Alice):

$$k = 2^{ab} \pmod{53} = 45^b \pmod{53} = 45^{19} \pmod{53} = 21 \in F_{53}.$$

Ovviamente l'esempio qui illustrato non è utilizzabile nella realtà, infatti sia il sistema crittografico classico di tipo Caesar, sia il campo $F_{53}$ per il sistema crittografico a chiave pubblica sono totalmente insicuri.

### 5.1.2 Il sistema crittografico di El Gamal

Sia $q$ un primo (grande) e $g$ un generatore di $F_q^*$. Sia $m$ il messaggio da inviare, con $m \in \mathbb{N}$, $0 \leq m \leq p-1$. Alice sceglie un intero $a$, tale che $0 \leq a \leq p-1$ e rende pubblico $g^a$, Bob manda ad Alice un messaggio crittato:

$$c = (g^b, m \cdot g^{ab}),$$

dove $b$ è un intero scelto da Bob, $0 \leq b \leq p-1$. Si dice che $m \cdot g^{ab}$ è il testo in chiaro "mascherato" e $g^b$ è l' "indizio". Alice, per decrittare, calcola:

$$(g^b)^{q-1-a} = g^{-ab}$$

e trova immediatamente $m$.

Il crittanalista si trova di fronte ad un problema equivalente al problema di Diffie - Hellman o a quello del logaritmo discreto.

### 5.1.3 Il gruppo delle curve ellittiche

Le curve ellittiche entrarono nella crittografia con il metodo di Lenstra per la fattorizzazione dei numeri interi (paragrafo 6.2.2). Successivamente e quasi contemporaneamente sia Koblitz [NK85] sia Miller [VM85] le utilizzarono per la costruzione di sistemi crittografici basati sullo schema dello scambio di chiavi di Diffie e Hellman e su quello di El Gamal.

La ragione fondamentale dell'interesse nelle curve ellittiche è dovuto al fatto che esse forniscono un'enorme quantità di gruppi abeliani finiti che, anche se con molti elementi, hanno un'operazione interna facilmente gestibile. In molti casi le curve ellittiche sono analoghe ai gruppi moltiplicativi dei campi sui quali sono definite, con il vantaggio di offrire una più vasta scelta. Altro grosso vantaggio offerto dalle curve ellittiche è che il problema del logaritmo discreto su di esse risulta essere più difficile da trattare che nei gruppi del tipo $F_q^*$ grazie a una struttura meno ricca di quella di campo: per esempio l'algoritmo degli indici (7.3), sul quale si basano alcuni degli algoritmi più veloci per il calcolo del logaritmo discreto, non è utilizzabile sulle curve ellittiche. Per questo motivo, i crittosistemi basati su curve ellittiche su $F_{2^n}$ richiedono valori relativamente piccoli di $n$ per essere sicuri.

#### La codifica dei messaggi

Sia $m$ il testo in chiaro, visto come intero positivo (dopo una prima codifica, come nell'esempio 4.1.1): si deve trovare un punto $P_m$ della curva ellittica $E$ in corrispondenza biunivoca con $m$.

Presentiamo di seguito alcuni semplici algoritmi probabilistici per codificare messaggi come punti di curve ellittiche; essi sono preferibili ad algoritmi deterministici perché generalmente più veloci.

Si consideri quindi una generica curva ellittica $E$ su $F_q$, dove $q = p^n$ e $p$ è un primo.

**(1)** Si supponga che $n$ sia pari e sia $0 \leq m < p^{n'}$, dove $n' = n/2$, per cui è possibile scriverlo nella forma:

$$m = m_0 + m_1 p + \cdots + m_{n'-1} p^{n'-1} \quad \text{con } 0 \leq m_j < p.$$

Si osservi che $F_{p^{n'}}$ può essere riguardato come spazio vettoriale su $F_p$; quindi si consideri una sua base

$$b_0, b_1, \ldots, b_{n'-1}$$

e si definisca

$$x(m) = m_0 b_0 + m_1 b_1 + \cdots + m_{n'-1} b_{n'-1}.$$

Sia $y(m) \in F_{p^n}$ la soluzione dell'equazione di Weierstrass che definisce la curva $E$ e e

$$P_m = (x(m), y(m)) \in E$$

è il punto cercato.

L'esistenza di $y(m)$ è garantita, infatti o $y(m) \in F_{p^{n'}}$ o in una sua estensione di grado 2, cioè $F_{p^n}$. Il grado dell'estensione è pari al grado del polinomio in $x$ dell'equazione di Weierstrass, in questo secondo caso irriducibile in $F_{p^{n'}}$.

**(2)** Sia $k$ un intero "grande" (l'errore si calcola con una probabilità di $1/2^k$, quindi $k = 30$ o $50$ è apprezzabile); sia $PT \in \mathbb{N}$ l'insieme dei possibili blocchi di tutti i possibili messaggi in chiaro già codificati in interi positivi, di modo tale che esista un $M \in \mathbb{N}$ tale che

$$m < M \quad \forall m \in PT \quad \text{e} \quad Mk < p^n = q;$$

si possono quindi scrivere gli interi compresi tra 1 e $Mk$ come interi di $n$ cifre in base $p$, ossia come polinomi di $F_p[x]^n$. Riguardando $F_q$ come spazio vettoriale su $F_p$ con base

$$b_0, b_1, \ldots, b_n$$

e sostituendo $x^i$ con $b_i$ negli elementi di $F_p[x]^n$, si ottengono elementi di $F_q$.

Così, dato $m$, si può ottenere, per ogni $j = 1, \ldots, k$, un $x \in F_q$ corrispondente a

$$\bar{x} = mk + j \in \mathbb{N}.$$

Si cerchi, se esiste, $y \in F_q$ tale che $y^2 = f(x)$ (per $f$ equazione di Weierstrass) e si ponga

$$P_m = (x, y).$$

Se non esiste tale $y$, si aumenta $\bar{x}$ di 1, si calcola il suo $x$ corrispondente e si riprova.

Se si trova

$$\bar{x} = mk + j \quad \text{per un certo } j = 1, \dots, k$$

tale che esiste un $y \in F_q$ per cui $y^2 = f(x)$, si ricava $m$ dal punto $(x, y)$ con

$$m = [(\bar{x} - 1)/k],$$

parte intera di $(\bar{x} - 1)/k$. La probabilità di avere $f(x)$ quadrato per qualche $y$ è di circa $1/2$, come illustrato nell'osservazione alla definizione 2.3.2.

**(3)** Sia $n = 1$, $p = q \equiv 3 \pmod 4$ e $0 \leq m < p/1000 - 1$. Si calcoli $x$ da $m$ concatenando tre cifre alla destra di $m$ (così $1000m \leq x < 1000(m + 1)$), fino a che non si trova un $y \in F_p$ tale che $f(x) = y^2$.

Se esiste tale $y$, il suo ordine divide $p-1$, e quindi l'ordine di $(f(x))$ divide $(p - 1)/2$, cioé

$$f(x)^{(p+1)/2} = f(x)$$

e quindi

$$y = f(x)^{(p+1)/4};$$

con $(p + 1)/4 \in \mathbb{N}$, perciò si pone

$$P_m := (x, f(x)^{(p+1)/4}) \in E$$

ed $m$ è calcolabile da $x$ eliminando le ultime tre cifre.

Analogamente a prima, la probabilità di avere $f(x)$ quadrato per qualche $y$ è di circa $1/2$.

**(4)** Sia $p = 2$ e $n \equiv 4 \pmod 6$, $b_0, b_1, \dots, b_{n-1}$ una base dello spazio vettoriale $F_{2^n}$ su $F_2$ e sia $y^2 + y = x^3$ l'equazione della generica curva ellittica $E$ su $F_2$.

Sia $m < 2^{n-10}$, per cui

$$m = m_0 + m_1 2 + \cdots + m_{n-11} 2^{n-11}$$

con $m_j \in \{0,1\}$ per $j = 0, \ldots, n-11$. Si cerchi

$$m_j \in \{0,1\} \quad \text{per} \quad j = n-10, \ldots, n-1$$

tale che $y^2 + y$ sia un cubo, con

$$y = m_0 + m_1 2 + \cdots + m_{n-1} 2^{n-1} \in F_{2^n}.$$

Il punto

$$P_m = (x, y) \in E$$

per

$$x = (y^2 + y)^{(2^n+2)/9}$$

è la codifica di $m$. Infatti se $(y^2+y)$ è un cubo, il suo ordine divide $(2^n-1)/3$, e quindi

$$\left[(y^2 + y)^{(2^n+2)/3}\right]^{1/3} = (y^2 + y)^{(2^n+2)/9}$$

e $(2^n + 2)/9 \in \mathbb{N}$, poiché 9 divide $2^n + 2$, infatti, se $n = 4 + k6$, si deve dimostrare che 9 divide $8 \cdot 2^{k6} + 1$, ossia

$$8 \cdot 2^{k6} \equiv -1 \pmod 9;$$

ma $6 = \varphi(9)$ e, per il teorema 2.1.2 di Eulero, $2^{k6} \equiv 1 \pmod 9$, da cui si ha $(2^n + 2)/9 \in \mathbb{N}$.

### Sistemi crittografici

**Definizione 5.1.1.** Data una curva ellittica $E$ su $F_q$ $(q = p^n)$ e $B, P \in E$, si dice *problema del logaritmo discreto su $E$ in base $B$*, la ricerca di $x \in \mathbb{Z}$ tale che $xB = P$, se tale $x$ esiste.

A questo punto si possono definire protocolli analoghi a quelli basati sul logaritmo discreto in $F_q$.

**Analogo del protocollo di Diffie Hellman.** Alice e Bob si mettono d'accordo su $F_q$ ed $E$ e rendono pubblico un elemento $B \in E$ (scelto in modo che il sottogruppo generato da $B$ sia abbastanza grande).

Alice sceglie $a \in \mathbb{Z}$ dello stesso ordine di grandezza di $q$ e calcola $aB \in E$ e lo rende pubblico. Analogamente fa Bob che rende pubblico $bB \in E$. Alice e Bob useranno come chiave il punto

$$P = abB \in E$$

o una espressione delle sue coordinate.

**Analogo del protocollo di El Gamal.** Siano $q \in \mathbb{N}$, $E$ una curva ellittica e $B \in E$ pubblicamente noti. Alice sceglie $a \in \mathbb{Z}$ e rende pubblico $aB$ (con $a$ segreto), Bob sceglie un $k \in \mathbb{Z}$ e invia

$$(kB, P_m + k(aB));$$

Alice calcola $akB$ dal primo elemento e, dal secondo, ottiene:

$$P_m = P_m + k(aB) - akB.$$

**La scelta della curva e di un suo punto**

Per il Teorema 2.4.8 di Menezes, Okamoto e Vanstone, la curva ellittica deve essere non supersingolare; per le implementazioni informatiche si consigliano curve definite su $F_{2^n}$. Per evitare attacchi con l'algoritmo di Silver Pohling e Hellmann è necessario trovare famiglie di curve per le quali è facilmente utilizzabile l'algoritmo di Schoof (nota a pag. 28).

Curve molto utilizzate sono:

$$E_n = \{(x, y) \in F_{2^n} \times F_{2^n} \mid y^2 + xy = x^3 + x^2 + 1\}$$

definita su $F_{2^n}$ di ordine

$$|E_n| = \left| \left( \frac{1 + \sqrt{-7}}{2} \right)^n - 1 \right|^2$$

e la

$$E(p) = \{(x, y) \in F_p \times F_p \mid y^2 + y = x^3\}$$

definita su $F_p$ (quindi non facilmente utilizzabile nella pratica) e che, per $p \equiv 1 \pmod 3$ ha ordine $p + 1 + a$, con $a$ definita da:

$$a^2 + 3b^2 = 4p, \quad a \equiv 1 \pmod 3 \quad \text{e} \quad b \equiv 0 \pmod 3.$$

## 5.2 Il sistema RSA

Il sistema crittografico RSA si basa sulla nota difficoltà di fattorizzare numeri interi, problema al quale si sono dedicati molti matematici negli ultimi tre secoli, tra i quali Fermat e Legendre, senza riuscire a trovare un algoritmo il cui tempo di esecuzione (in funzione del numero di cifre dell'intero da

fattorizzare) si avvicini al tempo oggi richiesto per la moltiplicazione di due interi. Il risultato migliore raggiunto fino ad oggi non si discosta molto dal risultato raggiunto da Legendre all'inizio del XIX secolo, argomento che rende affidabile il sistema.

Alice sceglie due primi $p_A$ e $q_A$, e calcola:

$$n_A = p_A q_A \quad \text{e} \quad \varphi(n_A) = (p_A - 1)(q_A - 1);$$

sceglie $e_A \in \mathbb{N}$, $1 < e_A < \varphi(n)$ e primo con $\varphi(n)$ (per evitare alcuni attacchi crittanalitici è consigliabile [K87] $\max(p, q) < e_A < \varphi(n)$), calcola

$$d_A = e_A^{-1} \pmod{\varphi(n_A)},$$

rende pubblica la chiave di crittazione

$$k_{E_A} = (n_A, e_A)$$

e tiene nascosta la chiave di decrittazione

$$k_{D_A} = (n_A, d_A).$$

Bob, che deve inviare ad Alice un messaggio $m$, calcola il crittotesto

$$c = E(c) = m^{e_A} \pmod{n_A}$$

e Alice può calcolare

$$m = D(c) = c^{d_A} \pmod{n_A}.$$

Questo sistema si basa sul seguente teorema:

**Teorema 5.2.1.** *$D(E(m)) = (E(D(m)) = m$.*

*Dimostrazione.* Per $n = p \cdot q$ prodotto di due numeri primi, si ha che

$$\varphi(n) = \varphi(p) \cdot \varphi(q) = (p-1)(q-1) = n - (p+q) + 1.$$

Omettendo l'indice A e scegliendo $e$ e $d$ come sopra, si può calcolare:

$$D(E(m)) \equiv (E(m))^d \equiv (m^e)^d \equiv m^{ed} \pmod{n}$$
$$E(D(m)) \equiv (D(m))^e \equiv (m^d)^e \equiv m^{ed} \pmod{n}$$

e

$$m^{ed} \equiv m^{k\varphi(n)+1} \pmod{n}$$

per qualche $k \in \mathbb{N}$. Per il Teorema di Eulero, se $p$ non divide $m$ segue che

$$m^{p-1} \equiv 1 \pmod{p};$$

quindi, da $(p-1)|\varphi(n)$ si ha

$$m^{k\varphi(n)+1} \equiv m \pmod{p}.$$

Se $m \equiv 0 \pmod{p}$, ossia se $p|m$, è banalmente vero che

$$m^{ed} \equiv m \pmod{p}.$$

Si può quindi dire che $\forall m \in \mathbb{N}$ e per $p$ e $q$ primi qualsiasi, è

$$m^{k\varphi(n)+1} \equiv m \pmod{p} \equiv m \pmod{q}$$

da cui segue che

$$m^{ed} \equiv m^{k\varphi(n)+1} \equiv m \pmod{n}.$$

Ciò prova che $D(E(m)) = (E(D(m)) = m$. □

L'RSA sfrutta la difficoltà computazionale di trovare $e_A^{-1} \pmod{\varphi(n_A)}$ senza conoscere $\varphi(n_A)$ e di trovare $\varphi(n_A)$ senza conoscere $p_A$ e $q_A$.

Se il messaggio è molto lungo, risulta necessario dividerlo in blocchi di $k$ lettere; analogamente il crittotesto, in fase di decrittazione dovrà essere diviso in blocchi di $l$ lettere. La scelta di $k$ e di $l$ andrà fatta in fase di costruzione del crittosistema, infatti, se si lavora con un alfabeto di $N$ lettere (ossia si considera ogni lettera come una cifra di un sistema di numerazione in base $N$), si sceglierà $n_A$ di modo che $N^k < n_A < N^l$, cosicché ogni blocco del testo in chiaro avrà come equivalente in base decimale un elemento di $\mathbb{Z}_{n_A}$ e ogni blocco del crittotesto $E(m)$ sarà un numero di $l$ (o meno) cifre in base $N$ univocamente determinato.

La costruzione dell'RSA richiede la ricerca di due primi sufficientemente grandi da scoraggiare attacchi crittanalitici. I test di primalità utilizzabili nella pratica sono algoritmi probabilistici (presentati nel paragrafo 6.1), quindi non deterministici, che possono lasciare il dubbio se i numeri effettivamente trovati siano primi o no, tuttavia essi richiedono molto meno tempo rispetto agli algoritmi più veloci sino ad ora proposti per rompere il sistema.

L'operazione $m^{e_A} \pmod{n_A}$ ed il suo inverso sono le più lunghe richieste dall'RSA: richiedono $O(h^3)$ operazioni binarie, (con il metodo delle quadrature ripetute 2.3.2) dove $h$ è il numero delle cifre degli interi elevati a potenza.

**Esempio 5.2.1.** Sia $N = 26$ (A corrisponde allo 0 e Z a 25), $k = 3$, $l = 4$. Bob manda il messaggio YES con la chiave di crittazione

$$(n_A, e_A) = (46927, 39423).$$

YES, in base 26, è rappresentato dal vettore $(24, 4, 18)$, che, in base 10, vale

$$24 \cdot 26^2 + 4 \cdot 26 + 18 = 16346.$$

Bob ottiene quindi il crittotesto

$$16346^{39423} \pmod{46927} = 21166 = 1 \cdot 26^3 + 5 \cdot 26^2 + 8 \cdot 26 + 2 = 1582$$

e $(1582)_{26}$ =BFIC.

Alice conosce

$$(n_A, d_A) = (46927, 26767)$$

e calcola

$$21166^{26767} \pmod{46927} = (16346)_{10} = \text{YES}.$$

Per evitare alcuni facili attacchi crittanalitici:

- $p$ e $q$ devono essere dei primi molto grandi (almeno 100 cifre ciascuno), e $N^k$ e $N^l$ devono avere almeno 200 cifre decimali;
- $p$ e $q$ non devono essere "vicini" perché ci sono algoritmi di fattorizzazione di un intero $n$ che iniziano i calcoli da $\sqrt{n}$;
- $(p-1)$ e $(q-1)$ devono avere MCD piccolo (comunque sono due numeri pari) e almeno un fattore primo "grande", infatti alcuni algoritmi si basano sulla "liscezza" di $(p-1)$ e $(q-1)$ (come l'algoritmo $p-1$ di Pollard illustrato nel paragrafo 6.2.1).

Nel Capitolo 6.2 verranno presentati gli algoritmi più noti per la fattorizzazione dei numeri interi.

## 5.3 Il sistema knapsack

Presentiamo un crittosistema presentato da Hellman e Merkle nel 1978 [HM78] basato su un noto problema $NP$; nel 1982 Shamir trovò un algoritmo polinomiale (in $k$) che lo risolve. Da quel momento in poi questo problema non fu quasi più utilizzato per la costruzione di crittosistemi, sebbene furono proposte delle sue varianti apparentemente sicure. Alcuni sostengono la necessità di studiare questo sistema crittografico e le sue varianti perché potrebbero tornare utili sia se i sistemi più diffusi oggigiorno (RSA, soprattutto, e logaritmo discreto) venissero rotti, sia perché potrebbero servire per una "composizione di sistemi crittografici" [SA96].

Il *problema dei pacchetti* (knapsack problem):

> Dato un insieme $\{v_1, \ldots, v_k, V\}$ di interi, dire se esiste un insieme
>
> $$\{\varepsilon_i | i = 1, \ldots, k;\ \varepsilon_i = 0, 1;\ \sum \varepsilon_i \cdot v_i = V\}$$
>
> e presentare tale insieme.

In caso di esistenza, tale insieme potrebbe non essere l'unica soluzione possibile.

Si dimostra che questo problema è di classe NP [Pa95].

Un caso particolare del problema dei pacchetti è il *problema dai pacchetti supercrescente*: si richiede che

$$v_i \geq \sum_{j=1}^{i-1} v_j.$$

Questo problema è molto più facile da risolvere del precedente e lo dimostra questo algoritmo:

**Algoritmo per risolvere il Problema dei pacchetti supercrescente.** Sia $W = V$ e $j = k$;

*a.* se $v_j > W$ si ponga $\varepsilon_j = 0$ e si vada al passo 3;

*b.* se $v_j \leq W$ si ponga $\varepsilon_j = 1$ e $W = W - v_j$;

*c.* sia $j = j - 1$;

*d.* se $j \neq 0$ e $W > 0$ si torni al passo 1.

Se si esce dal ciclo perché $W = 0$ allora il problema ha una soluzione e questa è data proprio dall'insieme degli $\varepsilon_j$ costruito; se invece si esce dal ciclo perché $j = 0$, ma $W > 0$, il problema non ha soluzione. Questo algoritmo dimostra anche che, se c'è una soluzione al problema dei pacchetti supercrescente, essa è unica.

**Esempio 5.3.1.** La 5-pla $\{2, 3, 7, 15, 31\}$ è supercrescente. Per $V = 24$ la soluzione al problema dei pacchetti è data dall'insieme ordinato $\{1, 0, 1, 1, 0\}$.

Per costruire il crittosistema, si divida il testo in blocchi i cui equivalenti numerici siano interi di k-bits (per esempio i blocchi di una lettera hanno equivalenti numerici da 0 a 25, ossia interi di 5-bit: da 00000 a 11001).

Alice sceglie (con un algoritmo generatore di numeri casuali) una k-pla supercrescente

$$(v_1, \ldots, v_k)$$

e due interi $n$ ed $a$ tali che

$$n > \sum_{i=1}^{k} v_i, \quad 0 < a < n \quad \text{e} \quad (n, a) = 1;$$

calcola quindi

$$b = a^{-1} \pmod{n} \quad (\text{di modo tale che } ab \equiv 1 \pmod{n}),$$

e la k-pla

$$(w_1, \ldots, w_k) \quad \text{con} \quad w_i = av_i \pmod{n}.$$

Alice rende pubblica la k-upla $(w_i)$. La chiave segreta di decrittazione sarà data da:

$$(v_1, \ldots, v_k, n, a, b).$$

Bob codifica il messaggio in chiaro $m$ in k-ple di numeri binari

$$(\varepsilon_k, \ldots, \varepsilon_1)$$

e calcola

$$c = E(m) = \sum_{i=1}^{k} \varepsilon_i w_i$$

e lo manda ad Alice, la quale, a sua volta, calcola

$$V = bc \pmod{n} = \sum_{i=1}^{k} \varepsilon_i w_i$$

per utilizzare la botola segreta, infatti:

$$bc = \sum_{i=1}^{k} \varepsilon_i b w_i = \sum_{i=1}^{k} \varepsilon_i v_i \pmod{n}.$$

Con l'algoritmo per risolvere il problema dei pacchetti supercrescente, da $V$ trova la k-upla

$$(\varepsilon_1, \dots, \varepsilon_k) = m.$$

Si osservi che $(\varepsilon_k, \dots, \varepsilon_1)$ è la k-upla cercata, dal momento che

$$V < n$$

e

$$\sum_{i=1}^{k} \varepsilon_i v_i \leq \sum_{i=1}^{k} v_i < n.$$

Il crittanalista ha a disposizione

$$c = \sum_{i=1}^{k} \varepsilon_i w_i,$$

con $(w_1, \dots, w_k)$ non supercrescente, perché la proprietà non si conserva da $(v_1, \dots, v_k)$ a $(w_1, \dots, w_k)$.

**Esempio 5.3.2.** Si scelgano la chiave di decrittazione $\{2, 3, 7, 15, 31\}$ (che è una 5-pla supercrescente), $n = 61$ e $a = 17$, dai quali si ottiene $b = 18$ e la chiave di crittazione $\{34, 51, 58, 11, 39\}$; sia

$$m = \text{WHY} \simeq (11000)(00111)(11000)$$

che si trasformano in:

$$(11000) \to 51 + 58 + 39 = 148$$
$$(00111) \to 34 + 51 + 58 = 143$$
$$(11000) \to 11 + 39 = 50$$

da cui si ottiene il crittotesto 148,143,50; per leggerlo è necessario moltiplicarlo per 18 modulo 61 ottenendo 41,12,46; utilizzando l'algoritmo precedente si riottiene il testo in chiaro.

*Osservazione.* Nel corso degli anni sono state presentate diverse varianti del crittosistema appena descritto ([K87], [SA96]), ma esse sono comunque ritenute insicure, perché si ritiene ([SA96]) che possano essere attaccate con opportune generalizzazioni dell'algoritmo di Shamir.

## 5.4 Un sistema crittografico con i codici correggibili

Il sistema crittografico che presentiamo qui brevemente, viene attualmente utilizzato per "comporlo" con quelli presentati precedentemente.

Sia $\mathcal{C}$ un $[n, k]$-codice lineare che può correggere fino a $t$ errori e con matrice di generatori $G$ di dimensione $k \times n$.

Alice sceglie come chiave pubblica una matrice $\bar{G}$ di dimensioni $k \times n$ e come chiave privata una coppia di matrici $(S, P)$ dove $S$ è non singolare, di dimensioni $k \times k$, detta "scompigliatrice", e $P$ è di dimensioni $n \times n$ tale che

$$\bar{G} = SGP$$

Bob vuole inviare ad Alice un messaggio $m$ di lunghezza $k$ (eventualmente lo suddivide in blocchi di lunghezza $k$), sceglie un vettore "sbagliato" $z$ di lunghezza $n$ e tale che $w(z) \leq t$ e calcola il crittotesto

$$c = E(m) = m\bar{G} + z$$

Alice decritta:

$$\bar{c} = cP^{-1} = (mSGP + z)P^{-1} = mSG + zP^{-1},$$

dove $w(z) \leq t$ e pertanto è decodificabile con il codice lineare generato da $G$. Si ottiene quindi $mS$ da cui si ottiene $m$ moltiplicandolo per $S^{-1}$.

# Parte III

# Metodi

# 6

# Primalità e fattorizzazione

In questo capitolo verranno presentati alcuni test di primalità e alcuni algoritmi per la fattorizzazione di interi. Osserviamo che un algoritmo che decompone in fattori primi un numero intero stabilisce anche se questo è primo o meno, ma i test di primalità sono generalmente più veloci di quelli di fattorizzazione.

## 6.1 Primalità

La maggior parte dei test di primalità consiste nella verifica di una condizione necessaria; insieme a questa verifica, si calcola la percentuale di casi in cui il numero soddisfa la condizione senza però essere primo; se un numero passa più verifiche, automaticamente si abbassa la probabilità che sia composto. Questo tipo di test viene anche detto *algoritmo probabilistico* (già illustrato nel paragrafo 1.5)

Un test di primalità, certamente banale e di tipo deterministico, è il calcolo del $MCD(m, n)$ per $m = 2, \ldots, [\sqrt{n}]$; se si trova un $m$ per cui $MCD(m, n) \neq 1$, allora $n$ è un numero composto. Questo algoritmo, in caso di risposta negativa, dà anche un fattore non banale del numero testato.

Un primo miglioramento di questo algoritmo è dato dal Piccolo Teorema di Fermat. Di seguito vedremo altri test che migliorano via via la richiesta di tempo.

### 6.1.1 Test di Fermat

Sia $n \in \mathbb{Z}$ il numero che si vuole testare; sia $[b]_n \in \mathbb{Z}_n^*$; si calcoli

$$d = MCD(b, n);$$

se $d \neq 1$, allora $n$ non è primo e il test si conclude con una risposta negativa; se $d = 1$, si calcoli

$$\overline{b} = b^{n-1} \pmod{n}$$

con il metodo dei quadrati ripetuti (teorema 2.3.2); se $\overline{b} = 1$, allora $n$ verifica la condizione necessaria data dal Piccolo Teorema di Fermat per $n$ primo.

**Definizione 6.1.1.** Siano $n \in \mathbb{N}$ dispari e $b \in \mathbb{N}$, tale che $MCD(n, b) = 1$; si dice che $n$ è *pseudoprimo in base b* (quindi "finge" di essere primo) se $b^{n-1} = 1 \pmod{n}$, e se $n$ è un numero composto.

**Esempio 6.1.1.** $n = 91$ è pseudoprimo in base 3:

$$3^{90} = 1 \pmod{91},$$

ma non è pseudoprimo in base 2:

$$2^{90} = 64 \pmod{91}.$$

**Definizione 6.1.2.** Si dice che $n \in \mathbb{N}$ non primo è un *numero di Carmichael* se passa il test di Fermat per ogni $[b]_n \in \mathbb{Z}_n^*$

**Teorema 6.1.1.** *Se $n \in \mathbb{N}$ dispari non passa il Test di Fermat per un certo $b \in \mathbb{N}$ tale che $1 \leq b \leq n-1$, allora $n$ fallisce il test per almeno metà degli interi positivi minori di $n$.*

*Dimostrazione.* Sia

$$S = \{b_1, \ldots, b_s | b_i \in \{1, \ldots, n-1\} \text{ e } b_i^{n-1} \equiv 1 \pmod{n}\},$$

che è l'insieme degli $s$ interi positivi minori di $n$ in base ai quali $n$ passa il testi di Fermat. Ovviamente $n$ non può essere pseudoprimo in base $bb_i$ con $b_i \in S$, altrimenti

$$(bb_i)^{n-1} \equiv 1 \pmod{n}$$

e

$$b^{n-1} = (bb_i)^{n-1}(b_i^{-1})^{n-1} \equiv 1 \pmod{n} \text{ (assurdo)};$$

quindi ci sono almeno $s$ distinti interi positivi minori di $n$ in base ai quali $n$ fallisce il Test di Fermat. Forzatamente, deve essere $s = (n-1)/2$. □

Quindi, se $n$ non è un numero di Carmichael, si ha una probabilità di $1/2$ che non passi il Test di Fermat; se $n$ passa il Test per $k$ diverse possibili basi, si ha una probabilità di $1/2^k$ che non sia primo.

### 6.1.2 Test di Jacobi

**Teorema 6.1.2.** *Sia $n$ un intero positivo; se $n$ è primo, allora per ogni intero positivo $a$, si ha:*

$$\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n} \tag{6.6.1}$$

*dove $\left(\frac{a}{n}\right)$ è il simbolo di Jacobi.*

*Dimostrazione.* Se $n|a$, allora i due membri della congruenza sono nulli; se $n \nmid a$, per il Piccolo Teorema di Fermat si ha

$$a^{(n-1)/2} \equiv \pm 1 \pmod{n}.$$

Sia $\mathbb{Z}_n^* =< g >$ e $a = g^j$; $j$ è pari se e solo se $a$ è un residuo quadratico modulo $n$, se e solo se $\left(\frac{a}{n}\right) = 1$; inoltre

$$a^{(n-1)/2} = g^{j(n-1)/2} = 1$$

se e solo se $n - 1|j(n-1)/2$, se e solo se $2|j$. □

**Definizione 6.1.3.** Ogni intero composto $n$ per cui vale la (6.6.1) viene detto *pseudoprimo di Eulero in base $a$.*

Si dimostra che, se $n$ non è primo, allora la probabilità che $n$ sia uno pseudoprimo in base $b$ è di $1/2$; se si prova per $k$ diversi $b$ che vale la (6.6.1) per un certo $n$, allora la probabilità che $n$ sia uno pseudoprimo di Eulero è di $1/2^k$.

*Osservazione.* Il miglioramento, rispetto all'algoritmo di Fermat, è che in questo caso non esistono analoghi dei numeri di Carmichael, quindi la probabilità indicata riguarda *tutti* gli interi.

### 6.1.3 Test di Miller - Rabin

Presentiamo brevemente il test di Miller - Rabin, che si basa sul seguente teorema:

**Teorema 6.1.3.** *Se $n \in \mathbb{N}$ è primo, allora*

$$\sqrt{1} \pmod{n} = \pm 1$$

**Definizione 6.1.4.** Sia $n$ un intero non primo; $b$ un intero postivo minore di $n$; sia

$$n - 1 = 2^s t;$$

se è soddisfatta una delle condizioni seguenti, si dice che $n$ è uno *pseudoprimo forte in base b*:

*a.* $b^t \equiv 1 \pmod{n}$

*b.* $2b^{2^r t} \equiv -1 \pmod{n}$.

**Teorema 6.1.4.** *Se $n$ è un intero dispari, non primo, allora $n$ è uno pseudoprimo forte in base $b$ ($0 \leq b < n$) per al massimo $1/4$ dei $b$.*

Quindi, se $n - 1 = 2^s t$ e si prova per $k$ diversi interi postivi $b < n$ che $n$ soddisfa una delle due condizioni esposte sopra, si ha una probabilità di $1/4^k$ che non sia primo.

## 6.2 Fattorizzazione di un intero

In questo paragrafo presentiamo due algoritmi per fattorizzare un intero positivo: il metodo $p - 1$ di Pollard e l'algoritmo di Lenstra che non è più veloce di altri, ma è certamente interessante perché fa uso della famiglia dei gruppi abeliani dei punti delle curve ellittiche, il cui numero di elementi fa sì che la ricerca di un fattore primo possa procedere con buone probabilità di successo.

Oggi sono a disposizione altri algoritmi molto efficaci come le variazioni del quadratic sieve ("setaccio" quadratico) proposto da Kraitchik, migliorato da Carl Pomerance e implementato efficacemente da Montgomery e Silverman [Sil87], o come il"setaccio" del campo dei numeri di John Pollard ([LLMP90] e [A91]), o, infine, come il metodo di Schnorr che fa uso delle approssimazioni diofantee, poi migliorate con delle riduzioni reticolari ([SC91] e [SCb91]).

In ogni caso tutti questi metodi non sono riusciti a rendere l'RSA un sistema insicuro, dovendo il progettista di crittosistemi evitare "solo" alcune trappole, peraltro ben note.

### 6.2.1 Il metodo $p - 1$ di Pollard

Questo metodo viene presentato perché su di esso si basa il metodo delle curve ellittiche proposto da Lenstra.

Sia $n \in \mathbb{N}$ dispari l'intero da fattorizzare; sia $p$ un suo fattore primo (per il momento sconosciuto).

**Definizione 6.2.1.** Si definisce la funzione:

$$M(k) = mcm\{1, \dots, k\},$$

che è il prodotto di tutti i numeri primi compresi tra 1 e $k$ presi alla massima potenza in cui appaiono nella fattorizzazione dei numeri compresi tra 1 e $k$.

*Osservazione.* L'algoritmo di Pollard non richiede l'uso esplicito di $M(k)$, ma un numero $M$ che sia multiplo di tutti gli interi minori di un intero $k$ (dalla cui scelta dipende l'esito dell'algoritmo); si potrebbe pertanto anche scegliere $M = k!$, ma $M(k)$ limita maggiormente la richiesta di spazio e di tempo.

L'algoritmo di Pollard si basa sulla possibilità che esista un $k \in \mathbb{N}$, con $k << p$, per il quale $p-1$ divida $M(k)$, infatti, se ciò non accadesse, il tempo e lo spazio richiesti dall'algoritmo risulterebbero non affrontabili. Illustriamo l'algoritmo:

*a.* sia $a \in \mathbb{N}$, con $2 \leq a \leq n-2$ (la scelta di $a = 2$ è indicata sia per l'implementazione informatica, sia perché 2 è primo sia con $n$, sia con $p$);

*b.* si calcoli $b = a^{M(k)} \pmod{n}$;

*c.* si calcoli $d = MCD(b-1, n)$ con l'algoritmo euclideo;

*d.* se $d = 1$ o $d = n$ si riparta dal passo 2 con un diverso $k$, se no $d$ è un divisore proprio di $n$.

Questo algoritmo funziona poiché, appena si trova un $k$ tale che $p-1$ divide $M(k)$, si ha:

$$b = a^{M(k)} \equiv 1 \pmod{p},$$

quindi $p$ divide $b-1$ e $p$ divide $d$. L'unica eccezione si verifica per

$$a^{M(k)} \equiv 1 \pmod{n},$$

per cui $d = n$.

Come si vede, questo algoritmo funziona bene se $p-1$ è un intero liscio (cioé con fattori primi "piccoli"), perché la ricerca di un $k$ adatto si ferma ai primi interi.

*Osservazione.* Il limite di questo algoritmo è che si basa sui gruppi $\mathbb{Z}_p^*$, che sono univocamente determinati dal gruppo abeliano finito $\mathbb{Z}_n$.

### 6.2.2 Fattorizzazione con le curve ellittiche

Questo metodo è stato presentato da H. W. Lenstra Jr. nel 1986 e si basa sul metodo $p-1$ di J. Pollard.

Si consideri la famiglia di curve ellittiche

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

dove la richiesta di avere il determinante $\Delta$ dell'equazione cubica primo con $n$ nasce dalla necessità di avere definite le curve ellittiche su $\mathbb{Z}_p$ per ogni fattore primo $p$ di $n$ (inoltre si osservi che se $d = MCD(\Delta, n) \neq 1$, $d$ stesso sarebbe un divisore di $n$).

*Osservazione.* Si verifica facilmente se $n$ è divisibile per 2 o per 3; se ciò non accade si può considerare una curva ellittica definita su un campo di caratteristica diversa da 2 o 3, e quindi l'equazione di Weierstrass che definisce la curva è riducibile al caso

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

dove la somma $P_3 = (x_3, y_3)$ di due punti $P_1$ e $P_2$ di coordinate $(x_1, y_1)$, e $(x_2, y_2)$ è data dalle equazioni:

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2$$
$$y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1,$$

che, per $P_1 = P_2$, diventano

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1$$
$$y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1.$$

**L'algoritmo di Lenstra** Sia $n \in \mathbb{N}$ l'intero dispari da fattorizzare, sia $E$ una curva ellittica definita su $\mathbb{Z}_n$ e sia $B$ un punto della curva. Dato un $k \in \mathbb{N}$ si cerchi di calcolare

$$M(k) \cdot B,$$

cosa non sempre possibile, dato che $\mathbb{Z}_n$ può fornire a $E$ dei punti di coordinate non invertibili, per i quali risulta impossibile eseguire le operazioni di somma. In particolare, avviene che $x_2 - x_1$ non è invertibile in $\mathbb{Z}_n$, se e solo se

$$MCD(x_2 - x_1, n) \neq 1.$$

Si potrebbe avere

$$MCD(x_2 - x_1, n) = n,$$

ossia $x_2 - x_1$ multiplo di $n$; in questo caso è necessario cambiare curva ellittica e rieseguire l'algoritmo.

Si vede facilmente che, a differenza dell'algoritmo di Pollard, ogniqualvolta i calcoli diventano troppo onerosi o ci si trova di fronte al caso

$$MCD(x_2 - x_1, n) = n,$$

è possibile cambiare la curva ellittica. La grande varietà di curve ellittiche a disposizione (corrispondente a tutte le possibili coppie $(a, b) \in \mathbb{Z}$) aumenta la probabilità di trovare un fattore primo proprio dell'intero positivo $n$.

# 7

# Calcolo del logaritmo discreto

Da alcuni decenni, sono stati proposti diversi algoritmi per il calcolo del logaritmo discreto. Per i progettisti di crittosistemi è molto importante stabilire quali proprietà possano accelerarli.

I primi risultati sono dovuti ai lavori di Pohling, Silver, Hellman e Pollard, che danno un algoritmo detto anche "baby step - giant step" (presentato nel paragrafo successivo), che hanno trovato un algoritmo che permette di calcolare il logaritmo discreto su qualsiasi gruppo $G$ in un tempo pari a $O(\sqrt{p_1})$, dove $p_1$ è il più grande primo che divide $|G|$. E' ovvio quindi che questo algoritmo è applicabile solo sui gruppi il cui ordine è liscio (ossia non divisibile per "grandi" primi). Il progettista di crittosistemi deve quindi evitare o campi $F_q$ tali che $q-1$ sia liscio o gruppi di curve ellittiche di ordine liscio. Per questo, nel caso delle curve ellittiche, è utilizzabile l'algoritmo di Shoof (ricordato a pagina 28) per il calcolo di $|E|$, che richiede $O(\log^9 q)$ operazioni binarie, ma non è facilmente praticabile, come già accennato nelle Osservazioni al Teorema 2.4.3 di Hasse.

Molto efficace è anche l'algoritmo degli indici (paragrafo 7.3), utilizzabile solo sui gruppi moltiplicativi di campi finiti $F_q$, e che impone, al progettista di crittosistemi, di scegliere un $q$ di almeno 1000 cifre. Infatti un crittanalista dovrebbe affrontare almeno $2^{100}$ operazioni binarie per rompere il sistema, contro le circa 2000 necessarie agli utilizzatori legali per utilizzare il sistema.

Un caso particolarmente interessante è l'utilizzo dei gruppi moltiplicativi dei campi finiti $F_{2^n}$ dato che sono facilmente implementabili [OD84], ma anche facilmente rompibili grazie ad un algoritmo di Coppersmith che calcola il logaritmo discreto su $F_{2^n}$ in

$$O(\exp c(n \log^2 n)^{1/3}))$$

operazioni gruppali [Cop84]. Una soluzione proposta riguarda l'uso dei numeri di Marsenne (ossia i primi della forma $2^n - 1$), che eviterebbero gli attacchi

dell'algoritmo di Pohling e Hellman, ma l'algoritmo degli indici non dipende sensibilmente dalla fattorizzazione di $q$; inoltre i numeri di Marsenne risultano essere troppo pochi e mal distribuiti in $\mathbb{N}$ per poter essere efficacemente utilizzati in fase di progettazione di crittosistemi.

Oggi si utilizzano curve ellittiche definite su campi $F_{2^n}$, perché godono delle stesse proprietà di implementazione dei campi $F_{2^n}$ stessi, ma non sono attaccabili con l'algoritmo di Coppersmith.

Si supponga di voler calcolare il logaritmo in un gruppo finito $G$ in base $g$ con ordine $n$. Per i gruppi generici il metodo più logico è quello di costruire una tavola con le $n$ potenze di $g$: si devono fare $n$ operazioni e la tavola occupa $O(n)$ spazio.

Un primo miglioramento (proposto da Shanks) richiede un tempo pari a $O(\sqrt{n}\log n)$ operazioni gruppali e uno spazio di $O(\sqrt{n})$ elementi: dato $m = [\sqrt{n}]$, sia

$$f : G \longrightarrow \{1, \ldots, n\}$$

una funzione iniettiva e si ccostruiscano i seguenti insiemi:

$$S = \{(i, f(ag^i)), i = 0, \ldots, m\} \tag{7.7.1}$$

$$T = \{(i, f(g^{mi})), i = 0, \ldots, m\} \tag{7.7.2}$$

Si cerchino gli elementi di $S$ e di $T$ che abbiano la seconda coordinata uguale, ossia tali che

$$ag^r = g^{mq}$$

per un certo $r$ tale che $0 \leq r \leq m$ ed un certo $q$ tale che $0 \leq q \leq m$, da cui

$$a = g^{mr-q}$$

e quindi

$$\log_g a = mr - q.$$

## 7.1 Algoritmo di Silver Pohling Hellman

Presentato nel 1978 da Pohling e Hellman [PH78] è stato scoperto indipendentemente anche da Roland Silver e da Richard Schroeppl con H.

Block. L'algoritmo presentato calcola il logaritmo discreto in $F_p$ per $p-1$ liscio (definizione 2.1.2).

Dati $y, b \in F_p^*$, si calcoli $x = \log_b y$, supponendo che $F_p^* =< b >$. Per ogni fattore primo $q_i$ di $p-1$ si calcolino le $q_i$-sime radici dell'unità di $F_p^*$:

$$r_{q_i,j} = b^{\frac{j(p-1)}{q_i}} \quad \text{per } j = 1, \ldots, q_i - 1$$

ottenendo la tabella $[r_{q_i,j}]$.

Sia

$$\prod_{q_i | p-1} q_i^{\alpha_i}$$

la fattorizzazione in primi di $p-1$. Se si calcola

$$x \pmod{q_i^{\alpha_i}}$$

per ogni divisore primo $q_i$ di $p-1$, col Teorema Cinese del Resto si trova $x$.

Si ponga $q = q_i$ e sia

$$x \equiv x_0 + x_1 q + \cdots + x_{\alpha-1} q^{\alpha-1} \pmod{q^\alpha},$$

con $0 \leq x_i < q$. Si esegua il seguente algoritmo per trovare $x_0$:

*a.* si calcoli $y^{(p-1)/q}$, che è una radice dell'unità da $y^{p-1} = 1$;

*b.* da $y = b^x$ e da $b^{p-1} = 1$ segue

$$y^{(p-1)/q} = b^{x_0(p-1)/q} = r_{q,x_0};$$

*c.* dalla tabella $[r_{q_i,j}]$ si cerchi un $j$ che soddisfi l'uguaglianza precedente e si ponga $x_0 = j$.

Per trovare $x_1$ si deve eseguire un algoritmo analogo:

*a.* si rimpiazzi $y$ con $y_1 := y/b^{x_0}$, da cui

$$\log y_1 = x - x_0 \equiv x_1 q^1 + \cdots + x_{\alpha-1} q^{\alpha-1} \pmod{q^\alpha};$$

*b.* si calcoli

$$y_1^{(p-1)/q^2},$$

che è una radice dell'unità da $y_1^{(p-1)/q} = 1$ (dal passo 2 precedente);

*c.* da $y_1 = b^{x-x_0}$ segue

$$y_1^{(p-1)/q^2} = b^{x-x_0(p-1)/q^2} = b^{x_1(p-1)/q} = r_{q,x_1};$$

*d.* dalla tabella $[r_{q_i,j}]$ si cerchi un $j$ che soddisfi la disuguaglianza precedente e si ponga $x_1 = j$.

A questo punto si procede induttivamente con il seguente algoritmo, per $i \geq 2$:

*a.* Sia

$$y_i = y/b^{x_0+x_1q+\cdots+x_{i-1}q^{i-1}}$$

per $y_0 = y$, da cui

$$\begin{aligned} \log y_i &= x - x_0 + x_1q + \cdots + x_{i-1}q^{i-1} \\ &\equiv x_iq^i + \cdots + x_{\alpha-1}q^{\alpha-1} \pmod{q^\alpha}; \end{aligned}$$

*b.* si calcoli

$$y_i^{(p-1)/q^{i+1}},$$

che è una radice dell'unità da

$$y_i^{(p-1)/q^i} = 1;$$

*c.* da

$$y_i^{(p-1)/q^{i+1}} = b^{x_i(p-1)/q}$$

segue

$$y_i^{(p-1)/q^{i+1}} = b^{x_i(p-1)/q} = r_{q,x_i};$$

*d.* dalla tabella $[r_{q_i,j}]$ si cerchi un $j$ che soddisfi la disuguaglianza precedente e si ponga $x_i = j$;

*e.* se $i \neq \alpha - 1$, si sostituisca $i$ con $i + 1$ e si torni al passo 1.

**Esempio 7.1.1.** Calcolare $\log_2 28$ in $F_{37}^*$, dove $F_{37}^* =< 2 >$. Si osservi che $37 - 1 = 2^2 3^2$.

Innanzitutto si calcolino le radici dell'unità; per $q = 2$ si ha:

$$2^{36/3} = 2^{18} \equiv 1 \pmod{37} \Longrightarrow r_{2,0} = 1 \text{ e } r_{2,1} = -1;$$

e per $q = 3$:

$$2^{12} \equiv 26 \pmod{37} \text{ e } 2^{24} \equiv 10 \pmod{37}$$

da cui segue

$$r_{3,0} = 1, r_{3,1} = 26 \text{ e } r_{3,2} = 10.$$

Quindi si esegue l'algoritmo; per $q = 2$ si ha:

$$x \equiv x_0 + 2x_1 \pmod{4}$$

e

$$28^{36/2} \equiv 1 \pmod{37}$$

da cui $x_0 = 0$;

$$28^{36/4} \equiv -1 \pmod{37}$$

da cui $x_1 = 1$ e quindi

$$x \equiv 2 \pmod{4};$$

per $q = 3$,

$$x \equiv x_0 + 3x_1 \pmod{9}$$

e

$$28^{36/3} \equiv 26 \pmod{37}$$

da cui $x_0 = 1$;

$$(28/2)^{36/9} = 14^4 \equiv 10 \pmod{37}$$

da cui $x_1 = 2$ e quindi

$$x \equiv 7 \pmod{9}.$$

Si osservi che l'unica soluzione possibile del sistema

$$\begin{cases} x \equiv 2 \pmod{4} \\ x \equiv 7 \pmod{9} \end{cases}$$

che sia minore o uguale di 36 è $x = 34$.

## 7.2 Algoritmo di Pollard

L'algoritmo di Pollard consente di calcolare il logaritmo discreto in $\mathbb{Z}_q$, quindi può essere ricondotto al problema generale nel calcolo di $x_i < q$, per

$$y_i^{(p-1)/q_{i-1}} = b^{x_i(p-1)/q}.$$

Si definisce una sequenza di elementi di $\mathbb{Z}_q$ dalla formula ricorsiva:

$$x_0 = 0$$

e

$$x_{i+1} = \begin{cases} yx_i \pmod q & \text{per } 0 < x_i < q/3 \\ x_i^2 \pmod q & \text{per } q/3 < x_i < 2q/3 \\ bx_i \pmod q & \text{per } 2q/3 < x_i < q \end{cases}$$

Quindi si ha

$$x_i \equiv y^{u_i} b^{v_i} \pmod q \tag{7.7.3}$$

dove $u_0 = v_0 = 0$ e

$$u_{i+1} = \begin{cases} u_i + 1 \pmod{q-1} & \text{per } 0 < x_i < q/3 \\ 2u_i \pmod{q-1} & \text{per } q/3 < x_i < 2q/3 \\ u_i \pmod{q-1} & \text{per } 2q/3 < x_i < q \end{cases}$$

e

$$v_{i+1} = \begin{cases} v_i \pmod{q-1} & \text{per } 0 < x_i < q/3 \\ 2v_i \pmod{q-1} & \text{per } q/3 < x_i < 2q/3 \\ v_i + 1 \pmod{q-1} & \text{per } 2q/3 < x_i < q \end{cases}$$

Seguendo Pollard, chiamiamo il minimo $i$ per cui $x_i = x_{2i}$ come *epatta* e la indichiamo con $e$. Questa costruzione richiama le sequenze casuali definite da una mappa su un insieme finito che genera una sequenza detta *a $\rho$-forma*; in queste sequenze, l'epatta si trova circa per $i = \sqrt{\frac{\pi^5 q}{288}}$ ([P78]). Si ha allora una sequenza semicasuale, e l'epatta si può discostare molto dal valore indicato.

Dalla (7.7.3) si ha

$$y^{u_e} b^{v_e} \equiv y^{u_{2e}} b^{v_{2e}} \pmod q$$

da cui

$$y^{u_e - u_{2e}} \equiv b^{v_{2e} - v_e} \pmod q;$$

si pongano $m$ ed $n$ di modo tale che

$$y^m \equiv b^n \pmod{q} \tag{7.7.4}$$

Sia ora

$$d = MCD(m, q-1) = \lambda m + \mu(q-1);$$

elevando la (7.7.4) alla $\lambda$ si ha

$$y^{\lambda m} \equiv b^{\lambda n} \pmod{q}$$

e, dal fatto che $|y|$ divide $q-1$, si ricava

$$y^d \equiv b^{\lambda n} \pmod{q}$$

e, da $y = b^x \pmod{q}$, segue che $\lambda n = dx$; per cui

$$y \equiv b^x \theta^i \pmod{q}$$

con $\theta \equiv b^{(q-1)/d}$ radice dell'unità. Una volta che si conoscono $\lambda$, $n$ e $d$, si può calcolare $x$.

Questo metodo può essere ripetuto per ogni gruppo abeliano e richiede $O(\sqrt{q})$ operazioni gruppali per trovare l'epatta.

E' possibile accelerare il processo abbreviando la ricerca dell'epatta a $O(\sqrt{q_1})$ operazioni gruppali, dove $q_1$ è il più grande primo che divide $q-1$.

Se $q - 1 = sq_1$, si può calcolare $Y = y^s$ e $B = b^s$, in modo da ridursi al problema

$$Y \equiv B^X \pmod{q_1}.$$

Calcolando l'epatta in $O(\sqrt{q_1})$ operazioni gruppali, si ha

$$Y^M \equiv B^N \pmod{q},$$

o

$$y^{sM} \equiv b^{sN} \pmod{q}$$

riconducendosi alla (7.7.4).

## 7.3 Algoritmo degli indici

Si tratta di un algoritmo molto importante dato che è stato utilizzato per attaccare con successo il logaritmo discreto in $F_q$.

Sia $p$ un primo e $q = p^n$; sia $F_q^*$ generato da $b$. Sia $f(x) \in F_p[x]$ polinomio irriducibile di grado $n$, per cui $F_q$ è isomorfo a $F_p[x]/f(x)$, cioé per ogni $a \in F_q$ è

$$a \simeq a(x) \in F_p[x]$$

con $\deg(a) \leq n-1$. I polinomi costanti di $F_p[x]$ sono gli elementi di $F_p \subseteq F_q$.

Sia $b' = b^{\frac{(q-1)}{(p-1)}}$, che è un generatore di $F_p^*$; questo consente di calcolare il logaritmo in base $b$ dei polinomi costanti (degli elementi di $F_p^*$) una volta che si è calcolato il logaritmo in base $b'$, dalla formula:

$$\log_b a = \frac{\log_{b'} a}{\log_{b'} b}$$

Se $p$ è piccolo, sarà facilmente costruibile una tabella dei $\log_{b'} a$; Nel caso $p = 2$ l'unico elemento non nullo è 1 e il calcolo della tabella risulta essere molto più veloce rispetto agli altri casi.

Inroduciamo la notazione $\text{ind}(a(x))$, che indica il logaritmo di $a(x)$ in base $b$ per $a(x) \simeq a \in F_q^*$.

L'algoritmo ha una fase di precomputazione: sia $B$ un sottoinsieme di $F_q$, detta *base*. Normalmente si sceglie $B$ insieme dei polinomi di $F_p[x]$ di grado minore di un certo $m < n$ e, per $h = |B|$, si fa in modo che $|F_p| = p < h < q$.

La parte di precomputazione consiste nel calcolare

$$\text{ind}(a(x)) \quad \text{per } a(x) \in B\text{:}$$

sia $t$ intero $1 \leq t \leq q-2$ e si calcoli $b^t \in F_q$, ossia, usando il metodo dei quadrati ripetuti, si calcoli:

$$c(x) \equiv b(x)^t \pmod{f(x)},$$

e sia

$$c(x) = \sum_{i=1}^{n-1} c_{n-i-1} x^i$$

e si cerchi di calcolare, dividendo via via $c(x)$ per gli $a(x) \in B$ scegliendo $\alpha_{c,a}$ come la più alta potenza di $a(x)$ che divide $c(x)$ in $F_p[x]$,

$$c(x) = c_0 \prod_{a \in B} a(x)^{\alpha_{c,a}} \tag{7.7.5}$$

(un buon algoritmo per trovare le $\alpha_{c,a}$ è stato presentato da Berlekamp e Knuth [KN69]); se alla fine non si ha un risultato del tipo (7.7.5), si deve ripartire dalla scelta di $t$.

Si supponga di aver trovato $c(x)$ che soddisfi la (7.7.5), allora è semplice calcolare:

$$\operatorname{ind}(c(x)) - \operatorname{ind}(c_0) \equiv \left[\sum_{a \in B} \alpha_{c,a} \operatorname{ind}(a(x))\right] \pmod{q-1} \tag{7.7.6}$$

dove è noto $\operatorname{ind}(c(x)) = t$, $\operatorname{ind}(c_0)$ dalla tabella sopra descritta e gli $\alpha_{c,a}$ perché ricavati esplicitamente. Non conoscendo gli $h$ valori di $\operatorname{ind}(a(x))$ con $a(x) \in B$, segue che la (7.7.6) è un'equazione in $\mathbb{Z}_{q-1}$ in $h$ incognite.

Si supponga quindi di trovare $h$ diversi $t$ per i quali sia soddisfatta la (7.7.5), e che si siano ottenute $h$ equazioni del tipo (7.7.6) con $h$ incognite, indipendenti tra di loro, che renderebbero quindi possibile la soluzione del sistema. In $F_p[x]$ ci sono $(p^m - p)/m$ polinomi monici irriducibili di grado $m$, per cui $h = |B|$ cresce molto rapidamente al crescere di $m$.

Ora, poiché si deve trovare una soluzione ad un sistema $h \times h$, gli $m$ (e quindi $h$) non devono essere troppo grandi; ma d'altro canto, se $m$ è piccolo è più difficile che un polinomio di grado minore di $n-1$ sia fattorizzabile dagli $a(x)$ di grado minore di $m$ (ossia è più facile che abbia un fattore irriducibile di grado maggiore di $m$).

Queste considerazioni dicono che è necessario fare parecchi tentativi per trovare le $t$ adatte; una scelta ottimale di $m$ dipende da $p$ ed $n$ e richiede una lunga analisi di probabilità e di tempo stimato. La precomputazione, quindi, termina dopo aver calcolato una tabella degli $\operatorname{ind}(a(x))$.

A questo punto sia $y \in F_q^*$ di cui si cerca $\log_b y$. Si scelga un $t'$ con $1 \leq t' < q-2$ e si calcoli

$$y_1 = yb^t,$$

ossia, per $y \simeq y(x) \in F_p[x]$,

$$y_1(x) \equiv y(x)b(x)^t \pmod{f(x)}.$$

Come prima cosa si deve valutare se

$$y(x) = y_0 \prod_{a \in B} a(x)^{\alpha_{y,a}}, \tag{7.7.7}$$

altrimenti si deve scegliere un diverso $t$ e ripartire da capo.

Se vale la (7.7.7) segue che

$$\mathrm{ind}(y(x)) = \mathrm{ind}(y_1(x)) - t$$

(dalla definizione di $y_1$ e $\mathrm{ind}(y_1) = \mathrm{ind}(y_0) \sum_{a \in B} \alpha_{y,a} \, \mathrm{ind}(a(x))$ di cui si conoscono tutti i termini del membro di destra.

*Osservazione.* Sono state proposte diverse varianti dell'algoritmo degli indici per migliorarlo e per adattarlo ai diversi tipi di campi, in particolare da Coppersmith per $F_{2^n}$. Alcune di queste varianti sono presentate in [OD84].

# Parte IV

# Altri protocolli

# 8

# Alcuni strumenti crittografici

## 8.1 Le funzioni Hash

**Definizione 8.1.1.** Sia $\Sigma$ un alfabeto, una funzione

$$h : \Sigma^* \longrightarrow \Sigma^r$$

(che manda quindi ogni stringa in stringhe di lunghezza fissata $r$) è detta *funzione hash*.

Le funzioni hash furono introdotte nei primi anni '50, per poter trasformare i messaggi in una limitata famiglia di valori (*valori hash*), in modo da verificare eventuali errori di trasmissione: un messaggio $m$ veniva inviato con il suo valore hash $h(m)$, il destinatario riceveva quindi la coppia $(m, h(m))$ e, da $m$ calcolava $h(m)$; se i due valori hash non combaciavano si era verificato un errore. Questo meccanismo non aveva scopi crittografici: un intercettatore attivo poteva riscrivere il messaggio e spedirlo con il valore hash ricalcolato.

Con l'avvento della crittografia a chiave pubblica le funzioni hash hanno acquisito maggiore importanza: usandole è possibile produrre firme digitali di lunghezza fissata che dipendano da tutto il messaggio e ne attestino l'autenticità utilizzando un valore $h(m)$ più semplice da utilizzare di $m$ perchè di lunghezza minore.

La definizione di funzione hash può essere modificata per ottenere altri strumenti crittografici:

**Definizione 8.1.2.** Siano $K$ un insieme delle chiavi e $\Sigma$ un alfabeto; una funzione

$$h : K \times \Sigma^* \longrightarrow \Sigma^r$$

è detta *funzione hash a senso unico con chiave* se:

*a.* dati $k$ e $m$ è facile calcolare $h(k, m)$;

*b.* dati $k$ e $h(k, m)$ è difficile calcolare $m$;

*c.* date molte coppie $(h(k, m), m)$ è difficile calcolare $k$;

*d.* dato $k$ è difficile trovare due valori $m$ ed $m'$ tali che

$$h(k, m) = h(k, m') \quad \text{con } m \neq m';$$

*e.* senza conoscere $k$ è difficile calcolare $h(k, m)$ per ogni $m$.

Il primo ed il terzo punto definiscono una funzione a senso unico con chiave; la quarta proprietà definisce una funzione immune da collisione.

Questa definizione è stata presentata da Berson, Gong e Lamos nel 1993, ma è stata accusata di presentare le funzioni hash a senso unico con chiave come combinazione di oggetti crittografici (funzioni hash e funzioni a senso unico) e non come oggetti "primitivi"; in ogni caso, quella presentata è una definizione comunemente accettata, anche se ne sono state proposte altre con l'obiettivo di mettere in risalto alcune proprietà di queste funzioni.

Non è negli scopi di questa tesi presentare esempi di funzioni hash: nel corso degli anni sono state presentate diverse proposte ma, a livello pratico, viene utilizzato lo standard americano proposto dal NIST (National Institute of Standards and Technology) in [FIPS180].

Più recentemente sono state proposte alcune classi di funzioni hash per utilizzarle in particolari contesti crittografici:

**Definizione 8.1.3.** Una classe di $H$ funzioni hash $h : A \longrightarrow B$ (qui non è richiesto che siano a chiave) è detta *universale*$_2$ se: $\forall m, m' \in A$ è

$$\delta_H(m, m') \leq |H|/|B|,$$

dove

$$\delta_H(m, m') = |\{h \in H \mid h(m) = h(m')\}|.$$

*Osservazione.* Queste classi di funzioni furono introdotte nel 1979 da Carter e Wegman nel tentativo di offrire un algoritmo per il deposito e l'utilizzo di chiavi in memorie comuni (Key Escrow) di classe di complessità temporale lineare e indipendente dall'input.

L'idea del Key Escrow richiede che le chiavi utilizzate per gli algoritmi crittografici vengano "spezzate" e che i vari pezzi siano depositati in alcune

memorie accessibili al pubblico, in modo da poterle "ricostruire" in caso di controversie legali o di indagini da parte dei servizi di sicurezza. In questi anni il NIST sta studiando uno standard per il Key Escrow con l'aiuto degli studiosi di crittografia di tutto il mondo.

Carter e Wegman definirono le classi fortemente universali.

**Definizione 8.1.4.** Una classe $H$ di funzioni hash che abbiano le proprietà della definizione precedente è detta *fortemente universale$_n$ ($SU_n$)* se, per qualunque $n$-upla di elementi distinti di $A$ $a_1, \ldots, a_n$ e per qualunque $n$-upla $b_1, \ldots, b_n$ di elementi non necessariamente distinti di $B$ si hanno $|H|/|B|^n$ funzioni di $SU_n$ che mandano $a_i$ in $b_i$.

Queste classi sono usate per le autenticazioni multiple.

## 8.2 Teoria della conoscenza zero

La Teoria della Complessità Conoscitiva è stata presentata da Goldwasser, Micali e Rackoff nel 1985 [GMR85], e calcola quanta conoscenza riesce a conquistare un partecipante con risorse computazionali limitate (polinomiali) ad una "conversazione" nella quale qualcuno cerca di dimostrare qualcosa. Nella pratica questa teoria è molto importante, perché si può presentare la necessità di dimostrare di possedere qualcosa (un algoritmo per fattorizzare i numeri interi in tempo polinomiale), senza che altri ne possano fare uso, sia in maniera diretta (venendo a conoscenza dell'algoritmo), sia in maniera indiretta (ponendo questioni al "dimostrante" la cui risposta possa essere riutilizzata). In questo caso non si considerano dimostrazioni formali (come quelle da testo di matematica), ma delle "prove" da mostrare per dimostrare che quanto asserito è vero.

Sono state proposte diverse varianti di questa teoria, sia dagli stessi Goldwasser, Micali e Rackoff, sia da altri, a seconda delle varie necessità, soprattutto relative alla costruzione di protocolli d'identificazione. Qui ci baseremo sull'articolo originale [GMR85].

**Definizione 8.2.1.** Si definisce una *macchina di Turing interattiva (MTI)*, una macchina di Turing con i seguenti elementi:

*a.* un nastro di input di sola lettura;

*b.* un nastro di lavoro;

*c.* un nastro con una serie infinita casuale di bits $(0, 1)$, detto *nastro casuale*, che può essere letto solo da sinistra verso destra; quando si chiede

alla MTI di lanciare una moneta, essa legge il bit successivo del nastro casuale a quello attualmente sul cursore; questo nastro è l'unica sorgente di casualità della MTI;

*d.* un nastro di comunicazione di sola lettura;

*e.* un nastro di comunicazione di sola scrittura.

**Definizione 8.2.2.** Due macchine di Turing interattive $A$ e $B$ formano una *coppia interattiva di MTI* se esse condividono il nastro di input e se il nastro di comunicazione di sola lettura di $A$ è il nastro di comunicazione di sola scrittura di $B$ e viceversa.

La coppia $(A, B)$ è da intendersi ordinata: $B$ inizia la computazione e le macchine lavorano a turno. Il *messaggio $i$-esimo $a_i$ di $A$* è la stringa $i$-esima che $A$ scrive sul suo nastro di sola scrittura al suo turno $i$-esimo; se la computazione totale richiede $n$ turni per macchina, si definisce *testo della computazione* la stringa

$$\{b_1, a_1, \ldots, b_n, a_n\}$$

($a_n = \sqcup$ se il lavoro termina all'$n$-esimo turno di $B$) e viene indicato anche con $(A, B)[x]$, dove $x$ è l'input iniziale.

**Definizione 8.2.3.** Sia $L \subseteq \{0,1\}^*$ un linguaggio, $(A, B)$ una coppia di MTI. Si dice che $(A, B)$ è un *sistema di prova interattivo* per $L$ se:

- $A$ ha potere computazionale infinito (e in particolare può risolvere problemi $NP$ in tempo polinomiale) e $B$ polinomiale;
- per ogni input $x \in L$ di $(A, B)$ di lunghezza $n$ sufficientemente grande, $B$ si ferma dopo un numero finito di turni allo stato "sì" con probabilità almeno $1 - 1/n^k$, per un certo intero $k$;
- per ogni MTI $A'$ diversa da $A$ e per ogni input $x \notin L$ di $(A', B)$, $B$ si ferma dopo un numero finito di turni allo stato "sì" con probabilità almeno $1/n^k$, per un certo intero $k$.

*Osservazione.* Se si considera $B$ come un "verificatore" allora si può vedere $A$ come una macchina che cerca di convincere $B$ del fatto che $x \in L$ o meno, senza per questo produrre una dimostrazione formale.

L'ultima proprietà della definizione precedente indica che $B$ non ha bisogno di conoscere la macchina con la quale interagisce per determinare se essa è un interlocutore autorizzato o meno; in questo secondo caso $B$ si accorge comunque del fatto che $A'$ cerca di dimostrare una cosa falsa.

**Esempio 8.2.1.** Sia $a \in \mathbb{Z}_m^*$ un residuo quadratico modulo $m$; sia

$$L = \{(m, x) \in \mathbb{Z}^2 | x \text{ non sia un residuo quadratico modulo } m\}.$$

$L$ è un linguaggio $coNP$: se $x$ è un residuo quadratico modulo $m$ si ha come certificato un $y$ tale che

$$y^2 \equiv x \pmod{m}.$$

Sia $A$ la macchina che dimostra a $B$ di saper distinguere i non residui quadratici modulo $m$.

$B$ scelga $n = |m|$ elementi casuali di $\mathbb{Z}_m^*$:

$$r_1, \ldots, r_n$$

e per ognuno di essi lanci una moneta; definisce

$$t_i = \begin{cases} r_i^2 \pmod{m} & \text{se viene testa e} \\ x r_i^2 \pmod{m} & \text{se viene croce;} \end{cases}$$

infine invia i $t_i$ ad $A$ che calcola quali sono i residui quadratici e descrive a $B$ la sequenza dei lanci della moneta, dimostrando così di riconoscere i non residui quadratici modulo $m$.

Se $(m, x) \notin L$, allora $x$ è un residuo quadratico modulo $m$ e quindi $A$ riceve unicamente una sequenza di residui quadratici e non può fare altro che tirare ad indovinare i possibili lanci: ogni lancio ha la probabilità di essere indovinato pari a $1/2$, quindi $A$ indovina la sequenza con una probabilità pari a $2^{-n}$.

**Definizione 8.2.4.** Sia $I$ un insieme finito di stringhe, $c$ una costante positiva; per ogni stringa $x \in I$ di lunghezza $n$, sia $\Pi_x$ una distribuzione probabilistica sulle stringhe di $n^c$ bits. Si definisce una *$I$-$c$-famiglia* l'insieme

$$\Pi = \{\Pi_x \text{ tale che } x \in I\}$$

(si possono omettere i riferimenti a $I$ e a $c$ se ne è assicurata l'esistenza).

Si dice *discernitore* un algoritmo probabilistico

$$D : I \longrightarrow \{0, 1\}$$

di classe di complessità temporale polinomiale.

Si osservi che le $I$-$c$-famiglie potrebbero essere dei testi di computazione $(A, B)[x]$.

**Definizione 8.2.5.** Siano $\Pi_1$ e $\Pi_2$ due $I$-$c$-famiglie, sia $p^D_{x,1}$ la probabilità che un discernitore $D$ dia risultato 1 per una stringa $x$ di lunghezza $|x|^c$ in $\Pi_{1,x}$; sia $p : \mathbb{N} \longrightarrow [0,1]$. Si dice che $\Pi_1$ e $\Pi_2$ *sono al massimo p-distinguibili* se per ogni discernitore $D$ esiste un intero $k$ tale

$$|p^D_{x,1} - p^D_{x,2}| < p(|x|) + \frac{1}{|x|^k}$$

per $|x|$ sufficientemente grandi.

Se $\Pi_1$ e $\Pi_2$ sono 0-distinguibili, vuol dire che sono uguali per computazioni polinomiali.

**Definizione 8.2.6.** Siano $(A, B)$ una coppia di MTI, $I$ l'insieme dei possibili input, $f : \mathbb{N} \longrightarrow \mathbb{N}$ una funzione non decrescente. Si dice che $A$ *comunica al massimo $f(n)$ bits di conoscenza a $B$* se esiste una macchina di Turing probabilistica polinomiale tale che gli $I$-insiemi

$$\Pi = \{\Pi_x | x \in I\} \quad \text{e} \quad (A, B)[\cdot]$$

siano al massimo $(1-2^{-f(n)})$-distinguibili; si dice che $A$ comunica al massimo $f(n)$ bits di conoscenza se per ogni MTI $B'$ con la quale costituisce una coppia di MTI, le comunica al massimo $f(n)$ bits di conoscenza.

**Esempio 8.2.2.** Nell'esempio precedente, $A$ comunica 0 bits di conoscenza a $B$, però un'altra MTI $B'$ potrebbe inviarle $t_i = i$, in modo da far calcolare alcuni particolari residui di cui potrebbe servirsi (e di conseguenza acquisire conoscenza).

**Definizione 8.2.7.** Sia $A$ una MTI interattiva, sia $L \in \{0,1\}^*$ un linguaggio che ammette un sistema di prova interattivo, sia $f : \mathbb{N} \longrightarrow \mathbb{N}$ una funzione non decrescente. Si dice che $L$ ha *complessità di conoscenza $f(n)$* se, restringendo gli input di $(A, B)$ ad $L$, per ogni MTI $B$ per la quale $(A, B)$ è un sistema di prova interattivo per $L$, $A$ comunica al massimo $f(n)$ bits di conoscenza. Per ogni $x \in L$ si ha la garanzia di poter generare un testo della computazione "corretto" con una probabilità di distribuzione al massimo $(1 - 2^{-f(n)})$.

I linguaggi di complessità di conoscenza $f(n)$ costituiscono una classe di complessità che si indica con

$$\text{KC}(f(n)).$$

In altre parole: se $L$ è un linguaggio di classe $\text{KC}(f(n))$ e se una MTI $B$ interagisce con un'altra MTI $A$, $B$ può verificare che $x \in L$, ma, rispetto ad una computazione polinomiale, il testo della computazione è irrilevante per ogni altro scopo, indipendentemente dal fatto che $A$ interagisca con $B$ o con altre MTI.

## 8.3 Numeri pseudocasuali

Abbiamo già visto in precedenza l'utilizzo di sequenze di numeri casuali (nell'Esempio 8.2.1), e lo vedremo ancora nei protocolli di identificazione presentati nel prossimo capitolo. I computer moderni non sono però in grado di generare sequenze di numeri veramente casuali: per fare questo sono necessari algoritmi generatori di sequenze pseudocasuali, che richiedono come input (detto anche *seme* $s$) uno o più numeri realmente casuali.

Le sequenze di numeri pseudocasuali, a loro volta, sono interessanti per altri utilizzi crittografici come, per esempio, la generazione di chiavi per il sistema crittografico della base monouso (Esempio 4.2.4), ma allora occorre definire un generatore di numeri pseudocasuali sicuro dal punto di vista crittografico.

Ci riferiremo a generatori di cifre binarie, dato che sono utilizzati principalmente per applicazioni informatiche, ma la definizione è facilmente generalizzabile.

**Definizione 8.3.1.** Un generatore di numeri binari pseudocasuali che richieda come input $n$ numeri realmente casuali (seme) e che generi in tempo polinomiale in $n$ una sequenza

$$x_1, x_2, \ldots, x_{n^k}$$

si dice *sicuro dal punto di vista crittografico* se, dato l'algoritmo e i primi $t$ numeri usciti, ma non il seme, è impossibile prevedere $x_{t+1}$ con probabilità maggiore di $1/2$. Si dice anche che questo algoritmo supera la *prova del bit successivo*.

*Osservazione.* Sono state presentate altre definizioni di generatori di numeri pseudocasuali sicuri dal punto di vista crittografico, in particolare, oltre alla prova del bit successivo, è giudicata affidabile la prova cosiddetta *statistica*, ma è stato dimostrato che esse sono equivalenti (Teorema di Yao, [WD88]).

Presenteremo ora, negli esempi che seguono, due algoritmi generatori di cifre binarie semicasuali; il primo è molto semplice e veloce e perciò molto utilizzato, anche se non risulta sicuro dal punto di vista crittografico; il secondo si basa sul problema dei residui quadratici e dà maggior sicurezza.

**Esempio 8.3.1.** Siano $n$ un intero positivo, che definisce il gruppo $\mathbb{Z}_n$ sul quale si lavora, due interi $a$ e $c$, detti *moltiplicatore* e *incremento* con la proprietà che $MCD(a, n) = 1$; sia $x_1$, con $0 \leq x_1 < n$, il seme dell'algoritmo; la sequenza di numeri viene definita dalla formula ricorsiva:

$$x_k = ax_{k-1} + c \pmod{n} \qquad \text{per ogni } k > 0.$$

**Esempio 8.3.2.** Siano $p$ e $q$ due primi tali che $p, q \equiv 3 \pmod 4$, sia $n = p \cdot q$ e sia $y_0$ un residuo quadratico modulo $n$; si definisce la sequenza di cifre binarie pseudocasuali $x_0, x_1, \ldots$ come:

$$x_i = \begin{cases} 0 & \text{per } y_i \text{ pari} \\ 1 & \text{per } y_i \text{ dispari} \end{cases}$$

dove

$$y_{i+1} = y_i^2 \pmod n.$$

A livello intuitivo, un generatore di numeri pseudocasuali sicuro dal punto di vista crittografico è equivalente ad una funzione a senso unico, dal momento che, data la sequenza generata, è computazionalmente impossibile calcolare il seme $s$. Per questo motivo sono stati proposti altri algoritmi che si basano su funzioni a senso unico, ed in particolare sul logaritmo discreto ([WD88]).

# 9

# Altri protocolli crittografici

## 9.1 Autenticazioni, identificazioni e firme

Se la posta elettronica è chiamata a sostituire il tradizionale supporto cartaceo, ci deve essere la possibilità di "firmare" un documento elettronico.

Inoltre, se una persona deve stabilire una comunicazione attraverso strumenti informatici con altri, deve anche accertarsi che il proprio interlocutore non sia un intruso.

In [FS86] vengono distinti diversi livelli di sicurezza:

**autenticazione** : Alice può provare a Bob di essere Alice, ma nessun altro può provare di essere Alice (questo caso è utile quando Alice e Bob collaborano e devono solamente essere in grado di riconoscere intrusi);

**identificazione** : Alice prova a Bob di essere Alice, ma Bob non può provare a nessun altro di essere lui stesso Alice (in altre parole Bob non riuscirebbe a dimostrare ad un eventuale giudice che l'autore di un certo messaggio è Alice, posto che lui stesso avrebbe potuto crearlo);

**firma** : Alice prova a Bob di essere Alice, ma Bob non può provare neanche a se stesso di essere Alice (in questo caso è dimostrabile ad un giudice che l'autore del messaggio è Alice).

Questi protocolli richiamano immediatamente altre necessità nate nell'era della comunicazione digitale ed a distanza, tra cui quella di provare anche in maniera "veloce" l'identità del proprio interlocutore in un ambito di conversazioni a distanza e di poter, quindi, recedere dal contatto.

Nel corso degli anni sono stati presentati altri protocolli di firma con diverse proprietà per risolvere i problemi che la crescente mole di comunicazioni

"sensibili" attraverso canali pubblici (internet) pone; pensiamo, a titolo di esempio, alle firme "innegabili" che possono essere verificate solo con l'aiuto del firmatario o le firme di gruppo.

I protocolli che presentiamo sono basati sulle funzioni a senso unico introdotte per la crittografia a chiave pubblica dal 1976 (RSA e logaritmo discreto). Per i protocolli di firma, autenticazione e identificazione basati sui sistemi crittografici a chiave privata, si preferisce la denominazione *MAC (Message Autentication Codes)* [BSP95].

Le nuove tecniche di firma fanno uso di due strumenti molto importanti, oltre ai protocolli crittografici: protocolli a conoscenza zero e funzioni hash.

Un altro concetto che introdurremo in questa parte è quello di *Centro fidato*. Indichiamo con Centro fidato un ente, riconosciuto dai partecipanti, che progetti il protocollo ed assegni le chiavi ai partecipanti, senza però divulgarle ad altri. Nei protocolli presentati nel capitolo 5 i partecipanti potevano scegliere da soli le proprie chiavi, ma la firma digitale deve essere riconosciuta da tutti, e in alcuni casi è regolamentata per legge. In Italia la legge che definisce la firma digitale per il commercio elettronico è del 1997, ma solo nella primavera del 1999 sarà attuata, perché in questi due anni sono state valutate le aziende che hanno richiesto il certificato di Centro fidato.

### 9.1.1 La firma digitale

Una delle proprietà comuni ai crittosistemi a chiave pubblica presentati nel capitolo 5 è che, se $E$ e $D$ sono rispettivamente gli algoritmi di crittazione e di decrittazione, si ha

$$D(E(m)) = m,$$

e abbiamo anche dimostrato che nei crittosistemi presentati vale la proprietà

$$E(D(m)) = m,$$

che permette di sviluppare un protocollo di firma digitale.

Una firma digitale deve essere *dipendente dal messaggio*, oltre che *dipendente dal firmatario*, di modo tale che un ricevente non sia in grado di modificare il messaggio prima di presentare la coppia messaggio-firma ad un giudice, o che nessuno possa "incollare" la firma ad un qualsiasi messaggio.

Se Alice vuole mandare un messaggio $m$ a Bob, lo "decritta" con la sua chiave privata e crea $D_A(m)$; per non far leggere il messaggio ad eventuali

intercettatori lo critta con la chiave pubblica di Bob e invia

$$E_B(D_A(m)).$$

Bob, usando la sua chiave privata ottiene $D_A(m)$ e, usando quella pubblica di Alice ottiene il messaggio $m = E_A(D_A(m))$. E' sicuro della sua provenienza dal momento che l'unico utente a poter creare il messaggio $D_A(M)$ non può che essere Alice.

Presentiamo, come esempio, il protocollo di firma di ElGamal. Quello di Rivest, Shamir e Adleman (RSA) è facilmente ricavabile da quanto esposto sopra.

### Il protocollo di ElGamal

Si lavori in $F_p$ con generatore $b$, dove $p$ è un numero primo; sia $m \in F_p$ il messaggio da firmare; ogni utilizzatore ha una chiave privata $x$ e una chiave pubblica

$$y = b^x \pmod{p}.$$

Il protocollo richiede tre passi: sia $y$ la chiave pubblica di Alice,

*a.* Alice sceglie $k \in \mathbb{N}$, $0 \leq k \leq p-1$ tale che $MCD(k, p-1) = 1$;

*b.* calcola

$$r = b^k \pmod{p};$$

*c.* sceglie $s$ di modo tale che

$$b^m = b^{xr} b^{ks} \pmod{p},$$

con $0 \leq s < p-1$.

*d.* invia la tripletta $m, r, s$.

Si verifica l'autenticità della firma verificando che

$$b^m = y^r r^s \pmod{p}$$

(come si vede dal terzo passo del protocollo di firma).

### Altri protocolli di firma

Nel corso degli anni, le esigenze pratiche degli utilizzatori di crittosistemi hanno imposto agli studiosi una ricerca sia per definire rigorosamente le nuove richieste sia per tradurre in proposte concrete le nuove definizioni. Di seguito presenteremo due semplici costruzioni.

**Firme innegabili.** Una *firma innegabile* è un tipo di firma che non può essere smentita dal firmatario nel tentativo di danneggiare altri partecipanti al protocollo (si pensi al commercio elettronico).

Il protocollo prevede che i due partecipanti collaborino per testare la validità della firma: eventualmente viene scoperto il firmatario che cerca di dare risposte fuorvianti nel corso del protocollo per cercare di dimostrare la falsità della sua firma.

Sia $G$ un gruppo moltiplicativo ciclico con generatore $g$ di ordine $p$ primo; il protocollo si basa sull'impossibilità computazionale di calcolare il logaritmo discreto degli elementi di $G$. Siano Alice e Bob i due partecipanti al protocollo e sia $m$ il messaggio che Alice invia a Bob.

Alice sceglie un intero positivo $a$ come chiave privata e una chiave pubblica $x = g^a$; calcola $z = m^a$ e invia a Bob la coppia $(m, z)$.

Se Bob vuole verificare la validità del messaggio ricevuto, sceglie $b, \bar{b} \in G$ e calcola

$$m^{\bar{b}a}(g^a)^b = m^{\bar{b}a}x^b;$$

Alice risponde inviando

$$(m^{a\bar{b}}x^b)^{-a} = m^{\bar{b}}g^b;$$

Bob verifica che la risposta ricevuta corrisponda proprio con $m^{\bar{b}}g^b$.

Si supponga che Alice invii

$$r_1 = m^{-a'a\bar{b}}g^b \neq m^{\bar{b}}g^b;$$

Bob allora le invia un altro elemento

$$m^{\bar{d}a}(g^a)^d$$

e verifica se anche

$$r_2 = m^{-a'a\bar{d}}g^d \neq m^{d'}g^d.$$

Se Alice cercasse di dimostrare che Bob ha ricevuto una firma falsa da parte di un intercettatore attivo, non eleverebbe entrambi i membri ricevuti alla $(-a)$, da cui si avrebbe

$$(r_1 g^{-b})^{\bar{d}} \neq (r_2 g^{-d})^{\bar{b}};$$

se Bob riceve veramente una firma falsa $m^{a'}$ e se Alice segue correttamente il protocollo, si ha

$$(r_1 g^{-b})^{\bar{d}} = (r_2 g^{-d})^{\bar{b}}.$$

Per verificare la validità di questo protocollo sono necessari due teoremi di cui diamo il risultato; per maggiori particolari, vedasi [CvA89].

**Teorema 9.1.1.** *Alice può dare una risposta valida ad una firma falsa con probabilità al massimo di $p^{-1}$.*

*Dimostrazione.* Ogni scelta $(b, \bar{b})$ da parte di Bob corrisponde a $p$ possibili coppie; se Bob riceve una firma falsa $m^{a'}$ da parte di Alice con $a \neq a'$ e invia $b$ e $\bar{b}$, Alice cerca di "inventare" una nuova coppia $(b', \bar{b'})$, sperando che

$$m^{a'\bar{b}} g^{ab} = m^{a'\bar{b'}} g^{ab'}$$

se e solo se

$$m^{a'(\bar{b}-\bar{b'})} = g^{a(b'-b)};$$

ma si avrebbe anche

$$m^{\bar{b}} g^{b} = m^{\bar{b'}} g^{b'},$$

se e solo se

$$m^{(\bar{b}-\bar{b'})} = g^{(b'-b)}$$

se e solo se $x = x'$; assurdo. □

**Teorema 9.1.2.** *Anche se Alice ha potere computazionale infinito, non ha una probabilità maggiore di $p^{-1}$ di impedire a Bob di accorgersi che sta dando risposte sbagliate ad una firma valida.*

*Dimostrazione.* Se Alice riesce a calcolare $b$ e $\bar{b}$, cercherà di rispondere in modo tale che

$$(r_1 g^{-b})^{\bar{d}} = (r_2 g^{-d})^{\bar{b}}$$

se e solo se

$$r_2 = (r_1^{1/\bar{b}} g^{-b/\bar{b}})^{\bar{d}} g^d.$$

Ma

$$r_1^{1/\bar{b}} g^{-b/\bar{b}}$$

può essere vista da Bob come costante nota. □

**Firme di gruppo.** Una *firma di gruppo* è un protocollo di firma al quale partecipa un gruppo di persone con queste caratteristiche:

*a.* ogni membro del gruppo può firmare messaggi a nome del gruppo;

*b.* solo i membri del gruppo possono firmare i messaggi;

*c.* il destinatario non può individuare il membro del gruppo che ha firmato;

*d.* in caso di controversia si può identificare il membro del gruppo che ha firmato il messaggio.

Per costruire un protocollo di questo tipo è necessaria la presenza di un Centro fidato che progetti il crittosistema.

Il sistema più banale si basa sulla pubblicazione, da parte del Centro fidato, delle chiavi pubbliche di tutti i membri del gruppo, non associate al rispettivo nome (conosciuto solo al Centro fidato). In caso di controversia il Centro fidato rivela il singolo al quale è associata la chiave pubblica utilizzata per la firma. Il problema principale di questo protocollo è la necessità di un membro esterno al gruppo (il Centro fidato) per tutto il periodo in cui si fa uso di questo tipo di firma.

Altri sistemi sono stati proposti in [CvH91]

## 9.2 Gli schemi a soglia

**Definizione 9.2.1.** Sia $m$ un messaggio che viene diviso in $n$ blocchi

$$m_1, \ldots, m_n$$

di modo tale che:

*a.* la conoscenza di $k$ o più pezzi rende $m$ facilmente computabile;

*b.* la conoscenza di $k-1$ pezzi, o meno, qualunque essi siano, lascia $m$ completamente indeterminato.

Questo schema è detto *schema a soglia* $(k, n)$.

L'utilizzo di tali sistemi può essere esemplificato nel seguente modo: si consideri una società che fa uso di assegni firmati digitalmente; se a ciascun procuratore viene data la firma digitale, il sistema gli consentirà un veloce accesso alle risorse della società , ma può anche facilmente approfittarne; se per firmare un assegno fosse richiesto l'intervento di tutti i procuratori (che potrebbero anche essere più di dieci, per esempio), il sistema sarebbe sicuro, a meno che tutti si trovassero d'accordo nel truffare la società, ma troppo lento per essere conveniente. La soluzione normalmente adottata è un sistema a soglia $(3, n)$, dove $n$ è il numero dei procuratori tra i quali dividere la chiave di firma $m$: ad ognuno di essi è fornita una tessera magnetica (smart card) con il blocco $m_i$ di modo tale che tre di esse possano generare (e poi distruggere) $m$. Questo sistema prevede anche che al direttore, per esempio, vengano consegnate 3 tessere e ai vice direttori 2 tessere ciascuno (di modo tale che bastino due persone per firmare un assegno).

### 9.2.1 Lo schema di Shamir

Una prima soluzione, molto semplice, è stata data da Shamir nel 1979 [SA79]: sia $p$ un primo maggiore di $m$ e di $n$ e si consideri un polinomio di grado $k-1$, $f(x) \in \mathbb{Z}_p$ di modo tale che il termine noto sia $m$, e si calcoli il valore di $f(x)$ (modulo $p$) per $n$ diversi valori di $x$: $m_1, \ldots, m_n$.

Poiché $\mathbb{Z}_p$ è un campo, in esso è possibile risolvere un sistema di $k$ equazioni che definiscono completamente $f(x)$, ma la conoscenza di $k-1$ coppie $(x_i, m_i)$ lasciano i coefficienti di $f$ (e quindi anche $m_0$) completamente indeterminati.

Sono stati presentati algoritmi per risolvere sistemi di $n$ equazioni lineari in $O(n \log^2 n)$ operazioni, però anche algoritmi più lenti sono utilizzabili senza inconvenienti.

Questo schema consente di aggiungere o cancellare facilmente altri blocchi $m_i$ in relazione all'avvicendamento di procuratori senza perdita di sicurezza; è possibile cambiare le $m_i$ evitando di dover cambiare ogni volta $m$ (procedura utile per aumentare la sicurezza nel caso, ad esempio, di perdite di tessere); come detto prima è possibile creare uno schema gerarchico dei detentori degli $m_i$.

## 9.3 Smart Cards

Una Smart Card è una tessera di plastica, di dimensioni standard, con un'unità di elementi elettronici, tra i quali una memoria e un microprocessore che regola la lettura e la scrittura su di essa. Si differenzia dalle normali carte a banda magnetica per il suo potere computazionale e per la memoria: ciò giustifica un nome diverso.

La Smart Card viene rilasciata da un Centro fidato alle persone che ne fanno richiesta; ogni carta serve, in generale, per un solo tipo di utilizzazione: in base a questa e al protocollo convenuto, il rapporto tra Centro fidato ed utente può concludersi al rilascio della carta o riprendere ogni volta che la si utilizza.

Le Smart Cards sono utili per protocolli di identificazione, di autenticazione e di firma; per il trasporto e l'utilizzo di chiavi per algoritmi crittografici; possono essere infine utilizzate per il commercio elettronico. Nella pratica odierna sono utilizzate per l'accesso a computer (o a settori della loro memoria), in sostituzione della password; o per l'utilizzo di apparecchi elettrici come i decodificatori televisivi. La Smart Card è nata alla fine degli anni Settanta ed è diventata subito molto popolare in Francia (le tessere telefoniche francesi sono delle Smart Cards, non delle tessere a banda magnetica come quelle italiane), fino ad imporsi con il passare degli anni anche in altri Paesi per lo sviluppo del commercio elettronico, per regolare il quale Russia e Stati Uniti hanno firmato digitalmente un accordo nel luglio 1998 con l'utilizzo di due computer collegati e due Smart Cards personalizzate.

Per i protocolli più comuni, nella memoria della Smart Card sono presenti dati relativi al proprietario (nome, cognome, ecc.) e alla carta (numero di serie, data di scadenza), codificate in modo tale da richiedere poco spazio, anche grazie all'utilizzo di funzioni hash. Gli schemi a conoscenza zero sono molto utili sia per questo scopo, sia per evitare che la carta venga copiata: infatti essa non rilascerebbe tutti i suoi dati (come i nostri bancomat, per esempio) ma solo una parte di essi, inutilizzabili da malintenzionati.

Il primo schema per Smart Cards con uso della Complessità di conoscenza (basato sull'RSA) è stato presentato da Fiat e Shamir nel 1986 [FS86]; nel 1988 Beth [BT88] propose il suo schema basato sul logaritmo discreto.

### 9.3.1 Lo schema di identificazione di Fiat e Shamir

Si supponga che Alice debba dimostrare la propria identità a Bob. Per

utilizzare questo protocollo è necessaria la presenza di un Centro fidato, che si occupa soltanto di:

*a.* rendere pubblico un intero positivo $n$, prodotto di due primi "grandi" ed un funzione $f : \Sigma^* \longrightarrow [0, n]$ casuale

*b.* creare una stringa $I$ per ogni utilizzatore (normalmente costruita combinando dati di quest'ultimo e della carta con stringhe casuali) e valutare $v_j = f(I, j)$ per $j = 1, \ldots, r$;

*c.* definire come $v_{j_1}, \ldots, v_{j_k}$ i primi $k$ elementi $v_j$ che siano residui quadratici modulo $n$ e valutare $s_{j_i} = v_{j_i}^{-1} \pmod{n}$ (normalmente $k = 30$ è più che sufficiente);

*d.* rilasciare la Smart Card con in memoria:

$$I, s_{j_1}, \ldots, s_{j_k}, j_1, \ldots, j_k.$$

Alice e Bob seguono il seguente protocollo, affinché Alice possa dimostrare la propria identità :

*a.* Alice invia $I$ e $j_1, \ldots, j_k$ a Bob;

*b.* Bob calcola $v_{j_i} = f(I, j_i)$ per $i = 1, \ldots, k$;

*c.* Alice sceglie un intero positivo casuale $r_i \in [0, n]$ e invia a Bob

$$x_i = r_i^2 \pmod{n} \quad \text{per } i = 1, \ldots, k;$$

*d.* Bob invia ad Alice un vettore $(e_{j_1}, \ldots, e_{j_k})$, con $e_{j_i} \in \{0, 1\}$;

*e.* Alice invia a Bob

$$y_i = r_i \prod_{i=1}^{k} e_{j_i} s_{j_i} \pmod{n};$$

*f.* Bob verifica che

$$x_i = y_i^2 \prod_{i=1}^{k} e_{j_i} v_{j_i} \pmod{n}$$

Se i passi $3-6$ vengono eseguiti $t$ volte, se Alice non conosce le $s_j$ e non riesce a calcolare in un tempo accettabile le radici quadrate di

$$\prod_{i=1}^{k} v_{j_i} e_{j_i} \pmod{n}$$

e se Bob segue correttamente il protocollo, Alice ha un probabilità di $2^{-kt}$ di trarre in inganno Bob. Infatti potrebbe cercare di indovinare i vettori $e_{j_i}$ inviando

$$x_i = r_i^2 \prod_{i=1}^{k} e_{j_i} v_{j_i} \pmod{n}$$

e, successivamente, $y_i = r_i$. Ogni volta che si seguono i passi $3-6$ del protocollo, Alice ha un probabilità di indovinare gli $e_{j_i}$ pari a $2^{-k}$.

Se Alice è sincera, Bob accetta la sua identità, dato che l'identità del passo 6 viene verificata.

Per un $k$ fissato ed un arbitrario $t$, Alice riesce a dimostrare di conoscere le $s_{j_i}$ senza dare alcun bit di conoscenza a Bob.

**Un miglioramento.** Sia $v \in \mathbb{N}$, per un qualunque intero positivo $j$, sia $J = f(I, j)$, dal quale il Centro fidato calcola

$$v_J = J^{1/v} \pmod{n}$$

e

$$s_J = v_J^{-1} \pmod{n},$$

che dà ad Alice, insieme a $v$ e $v_J$.

Alice invia a Bob un intero positivo $r$ minore di $n$, insieme a

$$x = r^v \pmod{n}$$

e ad $I$; Bob le invia un

$$e \in \{0, \ldots, v-1\};$$

Alice risponde con

$$y = r s_J^e \pmod{n}.$$

Bob verifica che

$$x = J^e \cdot y^v \pmod{n}.$$

Se Alice cerca di ingannare Bob senza conoscere $s_J$, può cercare di indovinare in anticipo $e$, con una probabilità pari ad $v^{-1}$. Si può scegliere $v$ di modo tale da ottenere con una o due ripetizioni del protocollo la stessa probabilità di sicurezza del protocollo di Fiat e Shamir, ed è proprio il risparmio di memoria (qui si devono memorizzare pochi $v_J$ e $s_J$) e di trasmissione (perché si diminuisce di $t$ volte la richiesta di bit da trasmettere) a parità di sicurezza sul quale si basa il miglioramento proposto.

### 9.3.2 Lo schema di Beth

Il Centro fidato sceglie un campo finito $F_q$, ed un suo elemento $\alpha$ e una funzione hash $h$; sceglie inoltre $m$ interi $x_1, \ldots, x_m$ e le

$$y_j = \alpha^{x_j} \quad \text{per } j = 1, \ldots, m.$$

Il Centro fidato calcola inoltre le

$$I_j = h(\text{dati della carta}, j),$$

sceglie un intero $k$ e calcola

$$r = \alpha^k$$

e determina $s_1, \ldots, s_m$, soluzioni delle:

$$rx_j + ks_j \equiv I_j \pmod{q-1}$$

Il Centro fidato rende pubblici: $q$, $\alpha$, $h$ e le $y_i$; registra sulla Smart Card $r$ e le $s_1, \ldots, s_m$.

Si suppone che Alice debba dimostrare la propria identità a Bob.

#### Il protocollo di autenticazione

*a.* Alice invia a Bob i dati della carta ed $r$;

*b.* Bob calcola

$$I_j = h(\text{dati della carta}, j)$$

e

$$\rho_j = y_j^r;$$

Alice e Bob ripetono i passi c - f per $i = 1, \ldots, h$;

*c.* Alice sceglie a caso $t_i \in \mathbb{Z}_{q-1}$ e calcola

$$z_i = r^{-t_i}$$

che invia a Bob;

*d.* Bob invia ad Alice una stringa casuale

$$(b_{i,j}) = (b_{i,1}, \ldots, b_{i,m}) \in X^m \subseteq \mathbb{Z}_{q-1};$$

*e.* Alice invia a Bob

$$u_i = t_i + \sum_{j=1}^{m} b_{i,j} s_j \pmod{q-1};$$

*f.* Bob calcola

$$v_i = \sum_{j=1}^{m} b_{i,j} I_j$$

e

$$\gamma_i = r^{u_i} z_i \prod_{j=1}^{m} \rho_j^{b_{i,j}} - \alpha^{v_i}.$$

Se $\gamma_i = 0$ il protocollo per $i$ è corretto.

Si verifica l'esattezza del protocollo dal momento in cui, se tutto è corretto:

$$\alpha^{v_i} = \alpha^{\sum_{j=1}^{m} b_{i,j}(r x_j + k s_j)} = \prod_{j=1}^{m} (\alpha^r)^{b_{i,j} x_j} \prod_{j=1}^{m} \left(\alpha^k\right)^{b_{i,j} s_j} =$$

$$= \left( \prod_{j=1}^{m} \rho_j^{b_{i,j}} \right) r^{u_i - t_i} = \left( \prod_{j=1}^{m} \rho_j^{b_{i,j}} \right) r^{u_i} z_i.$$

Se un impostore cerca di farsi credere Alice, sicuramente non ha le $s_j$ (non può né calcolare $k$ da $r$, né le $x_j$ dalle $y_j$ per il problema del logaritmo discreto), quindi, o, una volta ricevuta la stringa $b_{i,j}$, cerca gli $u_i$ tali che

$$r^{u_i} = \alpha^{v_i} \left( z_i \prod_{j=1}^{m} \rho_j^{b_{i,j}} \right)^{-1}$$

(che è sempre il problema del logaritmo discreto), o cerca di indovinare per tempo la stringa $b_{i,j}$ di modo da mandare delle $z - i$ opportune e, successivamente, le $u - i$, con una probabilità di riuscita pari a $1/|X|^{mh}$.

Questo è un protocollo a conoscenza zero.

# Bibliografia


[A91] Leonard M. Adleman, *Factoring Numbers Using Singular Integers,* Proceedings of the 23rd Annual ACM Symposium on Theory of Computation (STOC), ACM Press, New York, 1991, pp. 64-71.

[AHU74] Alfred V. Aho, John E. Hopcroft,Jeffrey D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison Wesley, Reading Massachusetts, 1974.

[BBCM87] Roberto Bevilacqua, Dario Bini, Milvio Capovani, Ornella Menchi, *Introduzione alla matematica computazionale,* Zanichelli, Bologna, 1987.

[BCLR81] Dario Bini, Milvio Capovani, Grazia Lotti, Francesco Romani, *Complessità numerica,* Bollati Boringhieri, Torino, 1981.

[BFS91] Thomas Beth, Markus Frisch, Gustavus J. Simmons, *Public-Key Cryptography. State of the Art and Future Directions*, LNCS 578, Springer Verlag, Berlino, 1991.

[BSP95] S. Bakhtiari, R. Safavi-Naini, J. Pieprzyk, *Keyed Hash Functions*, Cryptography, Policy and Algorithms, LNCS 1029, Springer Verlag, Berlino, 1995.

[BT88] Thomas Beth, *Efficient Zero-Knowledge Identification Scheme for Smart Cards*, in Advances in Cryptology - Eurocrypt '88, LNCS 330, Springer Verlag, Berlino, 1988.

[Cop84] Don Coppersmith, *Fast Evaluation of Logarithms in Fields of Characteristic 2*, IEEE Transactions on Information Theory, vol. IT-30, del 1984, pp. 587-594.

[CvA89] David Chaum, Hans von Antwapen, *Undeniable Signatures*, Advances in Cryptology - Crypto '89, LNCS 435, Springer Verlag, Berlino, 1989, pp. 212-216.

[CvH91] David Chaum, Eugene van Heyst, *Group Signatures*, Advances in Cryptology - Eurocrypt '91, LNCS 547, Springer Verlag, Berlino 1991, p. 257.

[DH76] Whitfield Diffie, Martin E. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory, vol. IT-22, no.6, Novembre 1976, pp. 644-654.

[DXS86] Cunsheng Ding, Guozhen Xiao, Weijuan Shan, *The Stability Theory of Stream Ciphers*, in Advances in Cryptology - Crypto '86, LNCS 263, Springer Verlag, Berlino, 1987, pp. 186-194.

[ELG85] Taher ElGamal, *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE Transactions on Information Theory, vol. IT-31, no.4, 1985, pp. 469-472.

[FIPS46] Federal Information Processing Standard, *Data Encryption Standard*, Publication 46, del gennaio 1977, National Institute of Standards and Technology (NIST), US Department of Commerce, Washington DC.

[FIPS81] Federal Information Processing Standard, *Data Encryption Standard, Modes of Operation*, Publication 81, del 1980, National Institute of Standards and Technology (NIST), US Department of Commerce, Washington DC.

[FIPS180] Federal Information Processing Standard, *Publication 180*, National Institute of Standards and Technology (NIST), US Department of Commerce, Washington DC.

[FS86] Amos Fiat, Adi Shamir, *How to Prove Yourself: Practical Solutions to Identification and Signature Problems*, Advances in Cryptology - Crypto '86, LNCS 263, Springer Verlag, Berlino, 1987, pp. 186-194.

[GMR85] Goldwasser, Micali, Rackoff, *The Knowledge Complexity if Interactive Proof Systems*, 17th ACM Symposium on Theory of Computation, ACM Press, New York, 1985.

[GO97] Oder Goldreich *On the Foundations of Modern Cryptography*, in Advances in Cryptology - Crypto '97, LNCS 1294, Springer Verlag, Berlino, 1997, pp. 46-74.

[H74] I. N. Herstein, *Topics in Algebra*, John Wiley & Sons, New York, 1974.

[HM78] Martin E. Hellman, R. C. Merkle, *Hiding Information and Signatures in Trapdoor Knapsacks*, IEEE Transactions on Information Theory, vol. IT-24, 1978, pp. 525-530.

[HU79] John E. Hopcroft, Jeffrey D. Ullman, *Introduction to Automata Theory, Languages and Computation* Addison Wesley, Reading Massachusetts, 1979.

[Hu87] Dale Husemoeller, *Elliptic Curves*, Springer, New York, 1987.

[IR72] Kenneth Ireland, Michael Rosen, *A Classical Introduction to Modern Number Theory*, Graduate Texts in Mathematics, vol. 84, Springer Verlag, Berlino, 1982.

[K85] Neal Koblitz, *Elliptic Curve Cryptosystems*, Mathematics of Computation, vol. 48 no.177, Gennaio 1987, pp. 203-209.

[K87] Neal Koblitz, *A Course in Number Theory and Cryptography*, Springer Verlag, Berlino, 1987.

[Kn69] D. E. Knuth, *The Art of Computer Programming, Vol II: Seminumerical Algorithms*, Addison-Wesley, Reading Massachusetts, 1969.

[LL96] Chae Hoon Lim, Pil Joong Lee, *Directed Signatures and Application to Threshold Cryptosystems*, Security Protocols, LNCS 1189, Springer Verlag, Berlino, 1996.

[LLMP90] A. K. Lenstra, H. W. Lenstra jr, M. S. Manasse, John M. Pollard, *The Number Field Sieve* Proceedings of the 22nd Annual ACM Symposium on Theory of Computation (STOC), ACM Press, New York, 1990, pp. 564-572.

[MV85] Victor S. Miller, *Use of Elliptic Curves in Cryptography*, Abstracts for Crypto '85, LNCS 218, Springer Verlag, Berlino, 1985

[OD84] A. M. Odlyzko, *Discrete Logarithms in Finite Fields and their Cryptographic Significance*, in Advances in Cryptology - Eurocrypt '84, LNCS 209, Springer Verlag, Berlino, 1985, pp. 224-314.

[P78] John M. Pollard, *Monte Carlo Methods for Index Computation* (mod $p$), Mathematics of Computation, vol. 32 no.143, Luglio 1978, pp. 918-924.

[Pa95] Christos H. Papadimitriou *Computational Complexity*, Addison-Wesley Publishing Company, Reading Massachusetts, 1995.

[PC90] Carl Pomerance (ed.) *Cryptology and Computational Number Theory*, American Mathematical Society, Providence Rhode Island, 1990.

[PH78] Stephen C. Pohling, Martin E. Hellman, *An Improved Algorithm for Computing Logarithms over $F_p$ and Its Cryptographic Significance*, IEEE Transactions on Information Theory, vol. IT-24, no.1 del Gennaio 1978, pp. 106-110.

[RSA78] R.L. Rivest, Adi Shamir, L. Adleman, *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, Communications of the ACM, vol. 21, no.2, Febbraio 1976, pp. 120-126.

[Sil87] R. D. Silverman, *The Multiple Polynomial Quadratic Sieve*, Mathematics of Computation, vol. 48, n.177 del 1987, pp. 329-339.

[SA96] Arto Salomaa *Public-key cryptography*, Springer, New York, 1996.

[SA79] Adi Shamir, *How to Share a Secret*, Communications of the ACM, vol. 22, no.11 del Novembre 1979, pp. 612-613.

[SA82] Adi Shamir, *A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem*, Proceedings of the 23rd Annual Symposium on the Foundations of Computer Science, del 1982, pp. 145-152.

[SC91] C. P. Schnorr, *Factoring integers and computing Discrete logarithms via Diophantine Approximation*, in Advances in Cryptology - Eurocrypt '91, Springer Verlag, 1991, pp. 141-146.

[SC91b] C. P. Schnorr, *Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems*, in FCT '91, LNCS 529, Springer Verlag, Berlino, 1991.

[SG79] S. G. Simmons, *Cryptology: the Mathematics of Secure Communication*, Mathematic Intelligencer, vol. 1, n.4 del 1979, pp. 233-246.

[SG79b] S. G. Simmons, *Symmetric and Asymmetric Encryption*, ACM Computing Surveys, vol. 11, n.4 del 1979, pp. 305-330.

[SR85] René Schoof, *Elliptic Curves over Finite Fields and the Computation of Square Roots* (mod $p$), Mathematics of Computation, vol. 44 del 1985, pp. 483-494.

[VG26] G. S. Vernam, *Cipher Printing Telegraph Systems for Secret Wire and Radio Telegraphic Communications*, Journal of AIEE, n.45 del 1926, pp. 109-115.

[WD88] Dominic Welsh, *Codes and Cryptography*, Oxford Science Publications, Oxford, 1988.