Teoria dei Codici e Crittografia

basato sull'omonimo corso
tenuto dal prof. Carlo Traverso

Anno Accademico 2011/12 - I semestre

Prima stesura: Marco Francischello
Revisione ed ampliamento: Oscar Papini

Indice

Prefazione

Queste note sono basate sul corso di *Teoria dei Codici e Crittografia* tenuto dal prof. Carlo Traverso durante il primo semestre dell'anno accademico 2011/2012. Esse sono il frutto di una sintesi tra

- appunti presi in classe (colgo l'occasione per ringraziare vivamente tutti coloro che li hanno forniti);
- libri consigliati:
  - per la parte di Teoria dei Codici, J. H. Van Lint, *Introduction to Coding Theory* (Springer, Terza ed., 1998);
  - per la parte di Crittografia, N. Koblitz, *A Course in Number Theory and Cryptography* (Springer, Seconda ed., 1994);
- altri testi e articoli;
- gli ottimi articoli di Wikipedia, ai quali rimando per ulteriori approfondimenti.

Queste note, a volte, si soffermano su alcuni argomenti più di quanto non si sia svolto durante il corso; sono tuttavia carenti di esempi, i quali si possono trovare facilmente nelle fonti già citate.

Naturalmente mi assumo ogni responsabilità per gli errori presenti nel testo.

Oscar Papini
`papini@mail.dm.unipi.it`

Ultima revisione: 17 dicembre 2013

Simboli e notazioni utilizzati

$\mathbb{K}, \mathbb{F}_q$	Campo, campo finito con q elementi
$\mathbb{Z}_n$	Anello degli interi modulo n; notazione breve per $\mathbb{Z}/(n)$
$\mathbb{P}^n(\mathbb{K})$	Spazio proiettivo n-dimensionale su $\mathbb{K}$
$\mathbb{A}^n(\mathbb{K})$	Spazio affine n-dimensionale su $\mathbb{K}$
$\mathcal{M}_{m\times n}(R)$	Spazio delle matrici $m \times n$ a coefficienti in R
R^*	Elementi invertibili dell'anello R
$\#A$	Cardinalità dell'insieme A
$\lfloor x \rfloor$	Parte intera di x

Nota. Le variabili indicate in corsivo (come x) indicano oggetti con una sola componente, quelle in grassetto (come $\mathbf{x}$) indicano oggetti con più componenti, come vettori o n-uple. Incognite o valori precisi sono generalmente indicati in minuscolo, mentre le indeterminate dei polinomi sono sempre indicate in maiuscolo.

Parte I

Teoria dei Codici

Capitolo 1

Introduzione

Che cos'è un *codice*? Pensandoci un secondo, la risposta a questa domanda non è affatto scontata. Potremmo definire codice *una qualsiasi rappresentazione di un insieme di informazioni*. Ad esempio, queste note sono scritte in italiano, e un lettore in grado di comprendere la lingua riesce ad estrarne l'informazione contenuta. Proviamo a formalizzare.

Definizione 1.1. Chiamiamo *alfabeto* un insieme finito i cui elementi sono detti *simboli*. Una *parola* è una sequenza finita di simboli dell'alfabeto. Un *codice*, infine, è un insieme di parole.

Non ci soffermeremo qui sul come estrarre informazioni da un codice, ci limiteremo a descrivere alcuni tipi di codice, nella fattispecie i *codici correttori*.

Definizione 1.2. Nel resto delle note, sceglieremo come alfabeto un campo finito $\mathbb{K}$, e un numero n che sarà la lunghezza di una parola. Pertanto, per noi, un *codice (a blocchi)* è un sottoinsieme di $\mathbb{K}^n$. Il numero n è detto anche *lunghezza* del codice.

Normalmente si sceglie $\mathbb{K} = \mathbb{F}_2$: tali codici sono detti *binari*.

1.1 Errori e distanze

Durante la trasmissione di un messaggio, potrebbero verificarsi degli errori. Questo dipende, per esempio, dalla qualità del canale attraverso cui avviene la trasmissione, oppure da situazioni specifiche del mezzo di trasmissione (interferenze elettromagnetiche,...). Siamo interessati a trovare codici che ci permettono di scoprire che una parola trasmessaci contiene un errore (*error detection*) ed eventualmente di correggerlo (*error correction*).

Supporremo sempre che ogni simbolo che trasmettiamo abbia probabilità $\alpha < \frac{1}{2}$ di essere sbagliato e che gli errori siano indipendenti. Una prima idea per la correzione potrebbe essere quella della decodifica di massima verosimiglianza (*maximum likelihood decoding*, o MLD): ricevuta una parola $\boldsymbol{w} = (a_1, \ldots, a_n)$ cerchiamo tra le parole del codice quella che ha maggiore probabilità di essere stata trasmessa, cioè quella con un minore numero di simboli diversi da $\boldsymbol{w}$. In caso vi siano più parole che soddisfino questa proprietà, possiamo chiedere la ritrasmissione del messaggio oppure scegliere una parola del codice arbitrariamente. Per formalizzare meglio questo concetto, introduciamo una definizione.

Definizione 1.3. Si dice *distanza di Hamming* tra due parole $\boldsymbol{w} = (a_1, \ldots, a_n)$ e $\boldsymbol{v} = (b_1, \ldots, b_n)$ la funzione $d(\boldsymbol{w}, \boldsymbol{v}) := \#\{i = 1, \ldots, n \mid a_i \neq b_i\}$; in altre parole, il numero di componenti diverse tra i due vettori. Chiamiamo *peso* di una parola $\boldsymbol{w}$ la sua distanza dalla parola nulla, ovvero $\mathrm{wt}(\boldsymbol{w}) := d(\boldsymbol{w}, \boldsymbol{0})$. Osserviamo che naturalmente $d(\boldsymbol{w}, \boldsymbol{v}) = \mathrm{wt}(\boldsymbol{w} - \boldsymbol{v})$. Chiamiamo *distanza* di un codice C il numero

$$d = \min_{\substack{\boldsymbol{v}, \boldsymbol{w} \in C \\ \boldsymbol{v} \neq \boldsymbol{w}}} \{d(\boldsymbol{w}, \boldsymbol{v})\}.$$

Cerchiamo di capire come è legata la distanza di un codice con il numero di errori rilevati e/o corretti.

Definizione 1.4. Sia C un codice, $\boldsymbol{v} \in C$ una parola inviata, e $\boldsymbol{w}$ la parola ricevuta. Si definisce *errore* tra $\boldsymbol{v}$ e $\boldsymbol{w}$ la parola

$$\boldsymbol{e} = \boldsymbol{w} - \boldsymbol{v}.$$

Diciamo che il codice *rileva* l'errore $\boldsymbol{e}$ se e solo se $\boldsymbol{v} + \boldsymbol{e} \notin C$ per ogni $\boldsymbol{v} \in C$. Diciamo che il codice *corregge* l'errore $\boldsymbol{e}$ se e solo se, per ogni $\boldsymbol{v}, \boldsymbol{w} \in C$ con $\boldsymbol{w} \neq \boldsymbol{v}$, si ha $d(\boldsymbol{w}, \boldsymbol{v} + \boldsymbol{e}) > d(\boldsymbol{v}, \boldsymbol{v} + \boldsymbol{e})$; in altre parole, se $\boldsymbol{v} + \boldsymbol{e}$ è più vicina a $\boldsymbol{v}$ che a qualsiasi altra parola di C.

Proposizione 1.5. *Sia C un codice di distanza d. Allora è in grado di rilevare al più $d - 1$ errori in una parola. Cioè: il codice rileva tutti gli errori $\boldsymbol{e}$ tali che $\mathrm{wt}(\boldsymbol{e}) \leqslant d - 1$, ed esiste un errore $\overline{\boldsymbol{e}}$ con $\mathrm{wt}(\overline{\boldsymbol{e}}) = d$ che non viene rilevato.*

Dimostrazione. Sia $\boldsymbol{e}$ un errore con $\mathrm{wt}(\boldsymbol{e}) \leqslant d - 1$ (e supponiamo $d \neq 1$). Per ogni $\boldsymbol{v} \in C$ si ha

$$d(\boldsymbol{v}, \boldsymbol{v} + \boldsymbol{e}) = \mathrm{wt}(\boldsymbol{v} + \boldsymbol{e} - \boldsymbol{v}) = \mathrm{wt}(\boldsymbol{e}) < d.$$

Dato che C ha distanza d, si ha che $\boldsymbol{v} + \boldsymbol{e} \notin C$ per ogni $\boldsymbol{v} \in C$. Quindi l'errore è rilevato.

D'altra parte esistono $\mathbf{c}_1, \mathbf{c}_2 \in C$ tali che $d(\mathbf{c}_1, \mathbf{c}_2) = d$, per definizione di d. L'errore $\overline{\mathbf{e}} := \mathbf{c}_1 - \mathbf{c}_2$ ha peso d, e $\mathbf{c}_2 + \overline{\mathbf{e}} = \mathbf{c}_1 \in C$: dunque $\overline{\mathbf{e}}$ non viene rilevato. □

Proposizione 1.6. *Sia C un codice di distanza d. Allora è in grado di correggere $\left\lfloor \frac{d-1}{2} \right\rfloor$ errori in una parola. Cioè: il codice corregge tutti gli errori $\mathbf{e}$ tali che $wt(\mathbf{e}) \leqslant \left\lfloor \frac{d-1}{2} \right\rfloor$.*

Dimostrazione. Sia $\mathbf{e}$ un errore con peso $wt(\mathbf{e}) \leqslant \left\lfloor \frac{d-1}{2} \right\rfloor$. Sia $\mathbf{v} \in C$ la parola trasmessa. Dobbiamo provare che $d(\mathbf{w}, \mathbf{v} + \mathbf{e}) > d(\mathbf{v}, \mathbf{v} + \mathbf{e})$ per ogni $\mathbf{w} \in C$ con $\mathbf{w} \neq \mathbf{v}$. Per la disuguaglianza triangolare (che è una delle proprietà di d), si ha

$$d(\mathbf{w}, \mathbf{v} + \mathbf{e}) + d(\mathbf{v} + \mathbf{e}, \mathbf{v}) \geqslant d(\mathbf{w}, \mathbf{v}),$$

e per ipotesi $d(\mathbf{v} + \mathbf{e}, \mathbf{v}) = wt(\mathbf{e}) \leqslant \frac{d-1}{2}$. Combinando queste disuguaglianze si ottiene

$$d(\mathbf{w}, \mathbf{v} + \mathbf{e}) + wt(\mathbf{e}) \geqslant d \geqslant 2wt(\mathbf{e}) + 1,$$

da cui $d(\mathbf{w}, \mathbf{v} + \mathbf{e}) \geqslant wt(\mathbf{e}) + 1 = d(\mathbf{v} + \mathbf{e}, \mathbf{v}) + 1$, da cui la tesi. □

1.2 Disuguaglianza di Hamming e codici perfetti

Sia $C \subseteq \mathbb{K}^n$ un codice; quante sono le parole che distano esattamente t, con $0 \leqslant t \leqslant n$, da una parola $\mathbf{v} \in \mathbb{K}^n$ fissata?

Per prima cosa, dobbiamo scegliere t simboli della parola: possiamo farlo in $\binom{n}{t}$ modi diversi. Per ciascuno di questi simboli abbiamo $\#\mathbb{K} - 1$ altri simboli tra cui scegliere. Quindi in totale abbiamo

$$\binom{n}{t}(\#\mathbb{K} - 1)^t$$

parole distanti esattamente t da una parola data. Di conseguenza, definendo le palle

$$B(\mathbf{v}, t) := \{\mathbf{w} \in \mathbb{K}^n \mid d(\mathbf{v}, \mathbf{w}) \leqslant t\},$$

abbiamo

$$\#B(\mathbf{v}, t) = \sum_{i=0}^{t} \binom{n}{i}(\#\mathbb{K} - 1)^i. \tag{1.1}$$

Osservazione. La cardinalità di una palla dipende solo dal raggio della palla e non dalla parola in cui è centrata.

Lemma 1.7. *Sia C un codice di distanza d, sia $t = \left\lfloor \frac{d-1}{2} \right\rfloor$, e siano $\mathbf{v}, \mathbf{w} \in C$ con $\mathbf{v} \neq \mathbf{w}$. Allora non esiste una parola $\mathbf{c} \in \mathbb{K}^n$ tale che $d(\mathbf{v}, \mathbf{c}) \leqslant t$ e $d(\mathbf{w}, \mathbf{c}) \leqslant t$.*

Dimostrazione. Se per assurdo esistessero $\boldsymbol{v}$, $\boldsymbol{w}$ e $\boldsymbol{c}$ come nell'enunciato, per la disuguaglianza triangolare si avrebbe

$$d(\boldsymbol{v}, \boldsymbol{w}) \leqslant d(\boldsymbol{v}, \boldsymbol{c}) + d(\boldsymbol{w}, \boldsymbol{c}) \leqslant t + t < d$$

contro la definizione di distanza. □

Teorema 1.8 (disuguaglianza di Hamming)**.** *Sia* $C \subseteq \mathbb{K}^n$ *un codice di distanza* d*, e* b *la cardinalità di una qualsiasi palla di raggio* $t = \left\lfloor \frac{d-1}{2} \right\rfloor$*. Allora*

$$b \cdot \#C \leqslant (\#\mathbb{K})^n.$$

Dimostrazione. Per il lemma 1.7, si ha che

$$\bigcup_{\boldsymbol{v} \in C} B(\boldsymbol{v}, t)$$

è un'unione disgiunta, e quindi ha cardinalità $b \cdot \#C$. La tesi segue facilmente. □

Definizione 1.9. Un codice si dice *perfetto* se la disuguaglianza di Hamming è un'uguaglianza.

In altre parole, un codice è perfetto se le palle centrate nei suoi elementi di raggio $\left\lfloor \frac{d-1}{2} \right\rfloor$ partizionano $\mathbb{K}^n$. Notiamo che un codice perfetto è sempre in grado di correggere un errore senza ambiguità (naturalmente non è detto che la correzione sia esatta).

Esempi di codici perfetti sono $\mathbb{K}^n$ stesso (che però ha distanza $d = 1$, e quindi non riesce a riconoscere errori) e un codice formato da una sola parola (la cui distanza non è definita, ma chiaramente ci si accorge se ci sono errori...). Questi sono detti *codici perfetti banali*. Vediamo, per esempio, quali sono gli unici codici perfetti non banali su $(\mathbb{F}_2)^n$ (la dimostrazione è omessa).

Teorema 1.10. *Se* $C \subseteq (\mathbb{F}_2)^n$ *è un codice perfetto non banale di distanza* d*, allora i possibili valori per* n *e* d *sono*

- n *dispari e* $d = n$*, e in tal caso* $C = \{(0, \dots, 0), (1, \dots, 1)\}$*;*
- $n = 2^r - 1$ *per qualche* $r \geqslant 2$ *e* $d = 3$*, i* codici di Hamming *(vedi sezione 2.4);*
- $n = 23$ *e* $d = 7$*.*

Capitolo 2

Codici lineari

Per condurre efficacemente le operazioni di rilevamento e correzione di errori (in una parola, di *decodifica*), risulta utile che il codice scelto per comunicare abbia una qualche struttura. Il primo esempio in tal senso sono i codici *lineari*.

Definizione 2.1. Un *codice lineare* di dimensione m è un sottospazio vettoriale m-dimensionale di $\mathbb{K}^n$.

Osservazione. Tutti i codici che presenteremo saranno in effetti dei codici lineari. Per questi codici notiamo che la distanza equivale al minimo peso delle parole non nulle del codice.

2.1 Matrice generatrice e matrice di parità

Dato che un codice lineare C è un sottospazio vettoriale di $\mathbb{K}^n$, diciamo di dimensione m, possiamo considerarne una base $\mathcal{B} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m)$. Costruiamo a questo punto la matrice $G \in \mathcal{M}_{m\times n}(\mathbb{K})$ che ha per *righe* gli elementi della base:

$$G = \left(\begin{array}{c} \boldsymbol{v}_1 \\ \hline \boldsymbol{v}_2 \\ \hline \vdots \\ \hline \boldsymbol{v}_m \end{array}\right).$$

Questa sarà chiamata *matrice generatrice* del codice. In effetti, ogni parola $\boldsymbol{w} \in C$ è ottenibile moltiplicando il vettore delle coordinate $\mathbf{u} = (a_1, \ldots, a_m) \in \mathbb{K}^m$ di $\boldsymbol{w}$ rispetto alla base $\mathcal{B}$ per la matrice G: $\boldsymbol{w} = \mathbf{u} \cdot G$.

Possiamo considerare anche una matrice $P \in \mathcal{M}_{n\times(n-m)}(\mathbb{K})$ che abbia per *colonne* dei generatori $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{n-m}$ di $C^\perp$. Tale matrice è detta *matrice di parità*

del codice:

$$P = \left(\begin{array}{c|c|c|c} & & & \\ w_1 & w_2 & \cdots & w_{n-m} \\ & & & \end{array} \right).$$

La seguente proposizione, la cui dimostrazione è omessa perché ovvia, mostra l'utilità della matrice di parità.

Proposizione 2.2. *Un elemento* $\mathbf{u} \in \mathbb{K}^n$ *appartiene a* C *se e solo se* $\mathbf{u} \cdot P = 0$.

Più in generale, chiameremo matrice di parità di un codice C una qualsiasi matrice P per cui valga il risultato della proposizione 2.2.

Definizione 2.3. Chiamiamo *sindrome* della parola $\mathbf{u}$ il vettore $\mathbf{s} := \mathbf{u} \cdot P \in \mathbb{K}^{n-m}$.

Quindi, per stabilire se una parola appartiene al codice, è sufficiente calcolarne la sindrome.

Dalla proposizione 2.2 segue immediatamente un corollario importante ai fini della decodifica.

Corollario 2.4. *Siano* $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{K}^n$, *e* $\mathbf{s}_1, \mathbf{s}_2$ *le rispettive sindromi. Allora* $\mathbf{s}_1 = \mathbf{s}_2$ *se e solo se* $\mathbf{v}_1 - \mathbf{v}_2 \in C$.

2.2 Distanza di un codice lineare

Esiste una disuguaglianza che lega tra loro la dimensione m del codice, la sua lunghezza n e la sua distanza d.

Lemma 2.5. *Sia* $C \subseteq \mathbb{K}^n$ *un codice lineare, e* P *una sua matrice di parità. Allora* C *ha distanza* d *se e solo se, scelte comunque* $d-1$ *righe di* P, *esse sono linearmente indipendenti, ed esistono almeno* d *righe di* P *linearmente dipendenti.*

Dimostrazione. Dalla proposizione 2.2, abbiamo $\mathbf{u} \in C$ se e solo se $\mathbf{u} \cdot P = 0$, cioè esplicitamente

$$\sum_{i=1}^{n} u_i \mathbf{p}_i = 0$$

dove $\mathbf{p}_i$ denota l'i-esima riga di P.

Dunque si ha una combinazione lineare con tutti i coefficienti non nulli tra h righe di P se e solo se esiste $\mathbf{u} \in C$ tale che $\mathrm{wt}(\mathbf{u}) = h$. Poiché per definizione d è il minimo peso non nullo di una parola del codice, ed è raggiunto da qualche parola, si ottiene che d è il minimo numero di righe richiesto per la dipendenza lineare, ed esistono effettivamente d righe linearmente dipendenti. □

Proposizione 2.6 (disuguaglianza di Singleton)**.** *Se* $C \subseteq \mathbb{K}^n$ *è un codice lineare di dimensione* m *e distanza* d*, si ha*

$$d - 1 \leqslant n - m.$$

Dimostrazione. Sia P matrice di parità del codice; dato che il suo rango massimo è $n - m$, non ci possono essere più di $n - m$ righe indipendenti. La tesi segue dunque da questa osservazione e dal lemma 2.5. □

2.3 Decodifica di un codice lineare

Abbiamo visto che la sindrome ci permette di rilevare un errore. In realtà essa ci fornisce informazioni anche su come correggerlo tramite MLD.

Fissiamo una sindrome $\mathbf{s} \in \mathbb{K}^{n-m}$, e consideriamo, tra le parole di $\mathbb{K}^n$ che hanno $\mathbf{s}$ come sindrome, quella con il peso minore. Denotiamo con $\boldsymbol{v}_\mathbf{s}$ tale parola.

Supponiamo che sia stata inviata la parola $\boldsymbol{v} \in C$, e sia stata ricevuta la parola $\boldsymbol{w} \in \mathbb{K}^n$. Calcoliamo la sindrome $\mathbf{s} = \boldsymbol{w} \cdot P$ e consideriamo la parola $\boldsymbol{v}_\mathbf{s}$. Per il corollario 2.4, $\boldsymbol{w} - \boldsymbol{v}_\mathbf{s} \in C$, e per minimalità del peso di $\boldsymbol{v}_\mathbf{s}$ tale parola è la più vicina a $\boldsymbol{w}$ tra quelle di C; è dunque la migliore candidata per la correzione dell'errore.

2.4 Codici di Hamming

Sia $\#\mathbb{K} = q$ (cioè $\mathbb{K} = \mathbb{F}_q$). Sia inoltre $r \geqslant 2$ un intero fissato. Quante sono le possibili stringhe non nulle di elementi di $\mathbb{K}$ lunghe esattamente r, in modo che siano a due a due linearmente indipendenti? È sufficiente controllare che non siano multiple a coppie; in totale saranno $q^r - 1$ (tutte le possibili stringhe di lunghezza r) diviso $q - 1$ (le righe multiple: sono tante quante gli elementi di $(\mathbb{F}_q)^*$). Fissiamo dunque $n = \frac{q^r-1}{q-1}$.

Definizione 2.7. Siano q, r, n come sopra. Il *codice di Hamming* di *ridondanza* r è il codice lineare in $(\mathbb{F}_q)^n$ la cui matrice di parità P ha per righe tutte e sole le r-uple descritte sopra, ed è quindi una matrice $\frac{q^r-1}{q-1} \times r$.

Un codice di Hamming su $\mathbb{F}_q$ di ridondanza r ha quindi lunghezza $\frac{q^r-1}{q-1}$ e dimensione $\frac{q^r-1}{q-1} - r$. Per esempio, per $q = 3$ e $r = 2$, la matrice di parità è

$$P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}.$$

Notiamo che la costruzione delle righe corrisponde a prendere $(\mathbb{F}_q)^r \setminus \{0\}$ e quozientare per la relazione di equivalenza

$$\boldsymbol{v_1} \sim \boldsymbol{v_2} \text{ se e solo se } \exists \lambda \in (\mathbb{F}_q)^* \text{ tale che } \boldsymbol{v}_1 = \lambda \boldsymbol{v}_2.$$

In altre parole, le righe della matrice di parità corrispondono ai punti dello spazio proiettivo $\mathbb{P}^{r-1}(\mathbb{F}_q)$.

Proposizione 2.8. *I codici di Hamming hanno distanza* $d = 3$*, e quindi permettono di correggere un errore. Inoltre sono codici perfetti.*

Dimostrazione. Dal lemma 2.5 e dalla definizione di codice di Hamming, segue che $d \geqslant 3$. D'altra parte è facile convincersi che esistono 3 righe linearmente dipendenti in P, quindi $d = 3$.

Per verificare che il codice è perfetto, calcoliamo intanto quanti elementi ha: $\#C = \#(\mathbb{F}_q)^{n-r} = q^{n-r}$. Per la formula (1.1), una palla di raggio $\left\lfloor \frac{d-1}{2} \right\rfloor = 1$ ha esattamente $b = 1 + n(q-1) = q^r$ elementi. Quindi

$$b \cdot \#C = q^r q^{n-r} = q^n = \#(\mathbb{F}_q)^n$$

e la tesi è dimostrata. □

Capitolo 3

Codici ciclici

Per questo capitolo supporremo sempre che la lunghezza n del codice e la caratteristica $\mathrm{char}(\mathbb{K})$ del campo siano coprime.

Definizione 3.1. Un codice *lineare* $C \subseteq \mathbb{K}^n$ è detto *ciclico* se per ogni $\mathbf{c} \in \mathbb{K}^n$ vale

$$(c_1, \ldots, c_n) \in C \Leftrightarrow (c_n, c_1, \ldots, c_{n-1}) \in C.$$

Per caratterizzare i codici ciclici sfruttiamo il fatto che lo spazio vettoriale $\mathbb{K}^n$ è isomorfo (come $\mathbb{K}$-spazio vettoriale) all'anello quoziente $\mathbb{K}[X]/(X^n-1)$, tramite la mappa

$$(c_0, \ldots, c_{n-1}) \longmapsto \left[c_0 + c_1 X + \cdots + c_{n-1} X^{n-1}\right]$$

dove indichiamo con $[p(X)]$ la classe del polinomio $p(X) \in \mathbb{K}[X]$ modulo $(X^n - 1)$. Nel seguito a volte identificheremo una parola $\mathbf{c} \in \mathbb{K}^n$ con la sua immagine $c(X)$, e viceversa.

Osservazione. L'applicazione $(c_0, \ldots, c_{n-1}) \longmapsto (c_{n-1}, c_0, \ldots, c_{n-2})$ corrisponde in $\mathbb{K}[X]/(X^n-1)$ alla moltiplicazione per il monomio X.

Lemma 3.2. *Un codice $C \in \mathbb{K}^n$ è ciclico se e solo se è un ideale in $\mathbb{K}[X]/(X^n-1)$.*

Dimostrazione. $\boxed{\Rightarrow}$ Dall'osservazione precedente, se C è ciclico e $[c(X)] \in C$, allora anche $[Xc(X)] \in C$, e per induzione anche $[X^n c(X)] \in C$, dunque $[a(X)c(X)] \in C$ per ogni $a \in \mathbb{K}[X]/(X^n-1)$. Dunque C è un ideale.
$\boxed{\Leftarrow}$ Sia $\mathbf{c} = (c_0, \ldots, c_{n-1}) \in C$, e sia $[c(X)]$ il polinomio corrispondente. Dato che C è un ideale, si ha $[Xc(X)] \in C$, e quindi $(c_{n-1}, c_0, \ldots, c_{n-2}) \in C$. Per il viceversa, basta moltiplicare per X^{n-1}. □

3.1 Polinomio generatore e matrice generatrice

Dato che $\mathbb{K}[X]/(X^n-1)$ è un anello a ideali principali, esisterà un polinomio $g(X)$ tale che $C = (g(X))$. Tale polinomio prende il nome di *polinomio generatore* del codice.

Proposizione 3.3. *Sia $g(X)$ polinomio generatore di un codice ciclico $C \subseteq \mathbb{K}^n$, e sia $n-m$ il suo grado. Allora $(g(X), Xg(X), \dots, X^{m-1}g(X))$ è una base di C come $\mathbb{K}$-spazio vettoriale, e quindi la dimensione di C è m.*

Dimostrazione. I gradi dei polinomi $g(X), Xg(X), \dots, X^{m-1}g(X)$ sono rispettivamente $n-m, n-m+1, \dots, n-1$, che sono tutti distinti. Quindi una loro combinazione lineare non viene mai ridotta modulo (X^n-1), né può portare a una cancellazione dei termini di testa. Questo prova che $g(X), Xg(X), \dots, X^{m-1}g(X)$ sono linearmente indipendenti.

D'altra parte, ogni parola $c(X) \in C$ si può scrivere come $a(X)g(X)$ per un opportuno polinomio $a(X)$, e possiamo supporre $\deg(a) < m$ (altrimenti, riduciamo $a(X)g(X)$ modulo X^n-1). Quindi

$$\begin{aligned} c(X) &= (a_0 + a_1X + \cdots + a_{m-1}X^{m-1})g(X) = \\ &= a_0g(X) + a_1Xg(X) + \cdots + a_{m-1}X^{m-1}g(X) \end{aligned}$$

e questo prova che $g(X), Xg(X), \dots, X^{m-1}g(X)$ generano C. □

Detto allora $g(X) = g_0 + g_1X + \cdots + g_{n-m}X^{n-m}$, e quindi la corrispondente parola $\mathbf{g} = (g_0, \dots, g_{n-m}, 0, \dots, 0) \in \mathbb{K}^n$, otteniamo la matrice generatrice $G \in \mathcal{M}_{m\times n}(\mathbb{K})$:

$$G = \begin{pmatrix} g_0 & \cdots & g_{n-m} & & & \\ & g_0 & \cdots & g_{n-m} & & \\ & & \ddots & \ddots & \ddots & \\ & & & g_0 & \cdots & g_{n-m} \end{pmatrix}.$$

Terminiamo con una proposizione che ci dice dove cercare polinomi generatori di codici ciclici.

Proposizione 3.4. *$g(X)$ è il polinomio generatore di un codice ciclico se e solo se divide X^n-1.*

Dimostrazione. $\boxed{\Rightarrow}$ Supponiamo che $g(X)$ sia il polinomio generatore di un codice ciclico. Con l'algoritmo di divisione euclidea si ha $X^n - 1 = g(X)f(X) + r(X)$, con $\deg(r) < \deg(g)$. Leggendo il tutto nell'anello $\mathbb{K}[X]/(X^n-1)$, abbiamo che $r(X) \in C$ e dunque, dovendo avere grado minore di $g(X)$, deve essere nullo.

$\boxed{\Leftarrow}$ Sia $g(X)$ un divisore di X^n-1. Vogliamo provare che è il polinomio generatore di $(g(X)) \subseteq {}^{\mathbb{K}[X]}/_{(X^n-1)}$. (Sarebbe ovvio se $(g(X)) \subseteq \mathbb{K}[X]$.) Sia per assurdo $h(X) \in (g(X))$ con $\deg(h) < \deg(g)$. Si ha che $g(X) \mid h(X)$ in ${}^{\mathbb{K}[X]}/_{(X^n-1)}$, cioè $g(X) \mid (h(X) + k(X)(X^n-1))$ in $\mathbb{K}[X]$. Ma per ipotesi $g(X) \mid (X^n-1)$, quindi $g(X) \mid h(X)$ in $\mathbb{K}[X]$, ma questo è assurdo perché $\deg(h) < \deg(g)$. □

3.2 Matrice di parità e decodifica

Sia C un codice ciclico con polinomio generatore $g(X)$ di grado $n-m$. Se viene inviata la parola $v(X)$ e la parola ricevuta è $w(X)$, abbiamo definito errore tra $v(X)$ e $w(X)$ la parola

$$e(X) = w(X) - v(X).$$

Vediamo come si traducono le sindromi nel linguaggio dei codici ciclici.

Definizione 3.5. Data una parola $w(X)$, definiamo *sindrome* di $w(X)$ la quantità $w(X) \mod g(X)$. Definiamo inoltre *matrice di parità* la matrice $P \in \mathcal{M}_{n\times(n-m)}(\mathbb{K})$ la cui i-esima riga è la parola di lunghezza $n-m$ corrispondente al polinomio $X^i \mod g(X)$, per $i = 0, \dots, n-1$.

Un paio di semplici conti mostrano che effettivamente sindrome e matrice di parità così definite rispettano le proprietà di sindrome e matrice di parità più generali dei codici lineari.

Nel caso di codici ciclici si può ottenere qualcosa in più. Abbiamo detto che $g(X)$ è un fattore di X^n-1. Sia allora $h(X)$ di grado m tale che $g(X)h(X) = X^n-1$. Si ha ovviamente che $g(X)h(X) = 0$ in ${}^{\mathbb{K}[X]}/_{(X^n-1)}$, cioè

$$g_0h_i + g_1h_{i-1} + \cdots + g_{n-m}h_{i-n+m} = 0 \qquad \forall i = 0, \dots, n-1.$$

Ciò significa che

$$H = \begin{pmatrix} & & & h_m \\ & & \iddots & \vdots \\ & h_m & \iddots & h_0 \\ h_m & \vdots & \iddots & \\ \vdots & h_0 & & \\ h_0 & & & \end{pmatrix}$$

è una matrice di parità per il codice C. Il polinomio $h(X)$ è chiamato *polinomio di controllo*, e si ha che $c(X) \in C$ se e solo se $c(X)h(X) = 0$ in ${}^{\mathbb{K}[X]}/_{(X^n-1)}$.

3.3 Codici ciclici ed estensioni di campi

Per quanto detto finora, non sembra esserci grande vantaggio nel lavorare con un codice ciclico rispetto a un semplice codice lineare.

Supponiamo che la matrice di parità di un codice lineare abbia un numero di colonne divisibile per ℓ, diciamo $P \in \mathcal{M}_{n\times k\ell}(\mathbb{F}_q)$. È possibile raggruppare ogni blocco di ℓ colonne in una sola estendendo il campo dei coefficienti. Infatti, ricordiamo che il campo $\mathbb{F}_{q^\ell}$ si può ottenere quozientando $\mathbb{F}_q[X]$ per un polinomio irriducibile di grado ℓ. Interpretando le ℓ colonne come i coefficienti della rappresentazione di un elemento di $\mathbb{F}_{q^\ell}$, si può vedere P come matrice in $\mathcal{M}_{n\times k}(\mathbb{F}_{q^\ell})$.

Scegliamo ora un codice ciclico su $\mathbb{F}_q$ di lunghezza $n = q^r - 1$. Come è noto, in tal caso le radici di $X^n - 1$ sono tutti e soli gli elementi di $(\mathbb{F}_{q^r})^*$. Dato che un polinomio generatore $g(X)$ divide $X^n - 1$, le sue radici sono alcuni elementi $\beta_1, \dots, \beta_t \in \mathbb{F}_{q^r}$.

Proposizione 3.6. *Sia $g(X) \in \mathbb{F}_q[X]$ polinomio generatore di un codice ciclico C di lunghezza $n = q^r - 1$, e siano $\beta_1, \dots, \beta_t \in \mathbb{F}_{q^r}$ le sue radici. (Ricordiamo che sono tutte distinte.) Allora*

$$C = \{c(X) \in \mathbb{F}_q[X] \mid c(\beta_1) = \cdots = c(\beta_t) = 0 \text{ in } \mathbb{F}_{q^r}\}.$$

Dimostrazione. $\boxed{\subseteq}$ È facile: $c(X) = g(X)a(X)$ per definizione di polinomio generatore, quindi $c(\beta_i) = g(\beta_i)a(\beta_i) = 0$.
$\boxed{\supseteq}$ Siano $f_i(X)$ i polinomi minimi su $\mathbb{F}_q$ dei β_i, e sia $c(X)$ che si annulla sui β_i. Chiaramente $\deg(f_i) \leqslant \deg(c)$, e dividendo si ottiene

$$c(X) = q(X)f_i(X) + s(X)$$

con $\deg(s) < \deg(f_i)$. Ma valutando in β_i si ottiene $s(\beta_i) = 0$, da cui $s(X) = 0$ per minimalità. Quindi ciascun f_i divide c, e di conseguenza anche $\operatorname{mcm}(f_1, \dots, f_t) = g$ lo divide. □

Quindi la valutazione può essere usata come decodifica; in effetti, la matrice di parità in tal caso risulta

$$P = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \beta_1 & \beta_2 & \cdots & \beta_t \\ \beta_1^2 & \beta_2^2 & \cdots & \beta_t^2 \\ \vdots & \vdots & \ddots & \vdots \\ \beta_1^{n-1} & \beta_2^{n-1} & \cdots & \beta_t^{n-1} \end{pmatrix}$$

e il prodotto $\mathbf{c} \cdot P$ corrisponde proprio alla valutazione $c(\beta_i)$ per $i = 1, \ldots, t$.

Possiamo dire qualcosa nel caso in cui n non sia proprio $q^r - 1$? In realtà sì, come ci dice il seguente risultato (la cui dimostrazione è omessa).

Proposizione 3.7. *Se n e q sono coprimi, allora $X^n - 1$ divide $X^{q^r-1} - 1$ per qualche r, e quindi $X^n - 1$ ha n zeri distinti in $\mathbb{F}_{q^r}$.*

In particolare, se n e q sono coprimi, allora n divide $q^r - 1$ per qualche r; siano $q^r - 1 = nb$, α elemento primitivo in $\mathbb{F}_{q^r}$, e $\beta = \alpha^b$. Allora tutti gli zeri di $X^n - 1$ (e quindi del polinomio generatore $g(X)$) sono potenze di β. In questo caso otteniamo un codice ciclico di lunghezza $(q^r - 1)/b$.

3.4 Codici di Hamming come codici ciclici

Ricordiamo che un codice di Hamming ha lunghezza $n = \frac{q^r-1}{q-1}$. Dalle osservazioni precedenti, scegliamo α elemento primitivo di $\mathbb{F}_{q^r}$, $\beta = \alpha^{q-1}$. In questo caso β è uno zero di $X^n - 1$, quindi il suo polinomio minimo (su $\mathbb{F}_q$) genera un codice ciclico di lunghezza n, la cui matrice di parità è

$$P = \begin{pmatrix} 1 \\ \beta \\ \beta^2 \\ \vdots \\ \beta^{n-1} \end{pmatrix}.$$

Questo è di fatto un codice di Hamming. In effetti, vale il seguente teorema (la cui dimostrazione è omessa).

Teorema 3.8. *Sia $n = \frac{q^r-1}{q-1}$, e β radice primitiva n-esima dell'unità in $\mathbb{F}_{q^r}$. Supponiamo che r e $q - 1$ siano coprimi. Allora il codice $\{c(X) \mid c(\beta) = 0\} \subseteq \mathbb{F}_q[X]$ è equivalente al codice di Hamming su $\mathbb{F}_q$ di ridondanza r.*

Capitolo 4

Codici BCH e RS

Proseguendo sulla stessa linea del capitolo precedente, studieremo due classi di codici ciclici: i codici BCH (chiamati così dal nome dei loro inventori Bose, Chaudhuri e Hocquenghem) e una loro generalizzazione, i codici RS (anche in questo caso, dal nome degli inventori Reed e Solomon).

4.1 Codici BCH

Definizione 4.1. Un *codice BCH* di *distanza designata* δ è un codice ciclico su $\mathbb{F}_q$ di lunghezza n (prima con q) il cui polinomio generatore è il minimo comune multiplo dei polinomi minimi su $\mathbb{F}_q$ di $\beta^\ell, \ldots, \beta^{\ell+\delta-2}$, per qualche ℓ, dove β indica una radice primitiva n-esima dell'unità in $\mathbb{F}_{q^r}$, con r dato dalla proposizione 3.7.

Il termine distanza designata trova giustificazione nella seguente proposizione.

Proposizione 4.2. *Un codice BCH di distanza designata δ ha distanza almeno pari a δ.*

Dimostrazione. Senza perdita di generalità, supponiamo $\ell = 1$. (La dimostrazione è analoga nel caso generale.) Supponiamo che $p(X) = b_1 X^{k_1} + \cdots + b_{\delta-1} X^{k_{\delta-1}}$ sia una parola del codice di peso minore o uguale a $\delta - 1$. Se $g(X)$ è il polinomio generatore del codice, $g(X) \mid p(X)$ e dunque abbiamo che:

$$\begin{pmatrix} \beta^{k_1} & \beta^{k_2} & \cdots & \beta^{k_{\delta-1}} \\ \beta^{2k_1} & \beta^{2k_2} & \cdots & \beta^{2k_{\delta-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \beta^{(\delta-1)k_1} & \beta^{(\delta-1)k_2} & \cdots & \beta^{(\delta-1)k_{\delta-1}} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{\delta-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Infatti, i vari prodotti riga per colonna corrispondono alla valutazione del polinomio $p(X)$ nelle radici $\beta, \dots, \beta^{\delta-1}$ di $g(X)$. Questa matrice, però, ricorda molto quella di Vandermonde; un conto mostra che il suo determinante è pari a

$$\left(\prod_{j=1}^{\delta-1} \beta^{k_j}\right)\left(\prod_{1\leqslant s<r\leqslant\delta-1} (\beta^{k_r} - \beta^{k_s})\right)$$

che è diverso da 0. Dunque $b_i = 0$. □

4.2 Decodifica di un codice BCH

Consideriamo un codice BCH di lunghezza $n = q^r - 1$ su $\mathbb{F}_q$ con distanza designata $\delta = 2t + 1$ e sia β un elemento primitivo di $\mathbb{F}_{q^r}$. Consideriamo una parola $c(X)$ del codice e la parola ricevuta

$$w(X) = w_0 + w_1 X + \cdots + w_{n-1} X^{n-1}.$$

Il polinomio d'errore è chiaramente $e(X) := w(X) - c(X) = e_0 + e_1 X + \cdots + e_{n-1} X^{n-1}$. Definiamo adesso

$$M := \{i \mid e_i \neq 0\},$$

ossia l'insieme delle posizioni dove sono occorsi gli errori, e indichiamo con $e = \#M$ il numero di errori. Ancora due definizioni importanti:

Definizione 4.3. Il *polinomio locatore d'errore* è

$$\sigma(Z) = \prod_{i\in M} (1 - \beta^i Z),$$

mentre il *polinomio valutatore d'errore* è

$$\omega(Z) = \sum_{i\in M} e_i \beta^i Z \prod_{j\in M\setminus\{i\}} (1 - \beta^j Z).$$

È chiaro che se noi riusciamo a trovare $\sigma(Z)$ e $\omega(Z)$, allora l'errore può essere corretto. Infatti c'è un errore in posizione i se e solo se $\sigma(\beta^{-i}) = 0$ e in tal caso l'errore è

$$e_i = -\frac{\omega(\beta^{-i})\beta^i}{\sigma'(\beta^{-i})}.$$

Naturalmente d'ora in avanti supporremo che $e \leqslant t$, altrimenti non siamo in grado di correggere errori. Si osservi che

$$\frac{\omega(Z)}{\sigma(Z)} = \sum_{i \in M} \frac{e_i \beta^i Z}{1 - \beta^i Z} = \sum_{i \in M} e_i \sum_{l=1}^{\infty} (\beta^i Z)^l = \\ = \sum_{l=1}^{\infty} Z^l \sum_{i \in M} e_i \beta^{li} = \sum_{l=1}^{\infty} Z^l e(\beta^l),$$

dove tutti i calcoli sono serie formali di potenze su $\mathbb{F}_{q^r}$. Notiamo che per $1 \leqslant l \leqslant 2t$ si ha $e(\beta^l) = w(\beta^l)$: infatti $e(\beta^l) = w(\beta^l) - c(\beta^l)$, e $c(X) = g(X)a(X)$, con g polinomio generatore, dunque $c(\beta^l) = 0$ per $l = 1, \ldots, 2t$. In altre parole, il ricevente conosce i primi $2t$ coefficienti di $\sum Z^l e(\beta^l)$: il rapporto $\omega(Z)/\sigma(Z)$ è noto modulo Z^{2t}.

Il ricevente, a questo punto, può determinare i polinomi $\sigma(Z)$ e $\omega(Z)$ tali che

1. il grado di ω non superi quello di σ;
2. il grado si σ sia il più piccolo possibile;
3. valga la condizione

$$\frac{\omega(Z)}{\sigma(Z)} \equiv \sum_{l=1}^{2t} Z^l w(\beta^l) \pmod{Z^{2t}}. \tag{4.1}$$

Sia $S_l := w(\beta^l)$ per $l = 1, \ldots, 2t$ e $\sigma(Z) = \sum_{i=0}^{e} \sigma_i Z^i$. Allora

$$\omega(Z) \equiv \left(\sum_{l=1}^{2t} S_l Z^l \right) \left(\sum_{i=0}^{e} \sigma_i Z^i \right) = \sum_k Z^k \left(\sum_{i+l=k} S_l \sigma_i \right) \pmod{Z^{2t}}.$$

Dato che $\omega(Z)$ ha grado al più e, si ha

$$\sum_{i+l=k} S_l \sigma_i = 0 \quad \text{per } e+1 \leqslant k \leqslant 2t.$$

Questo è un sistema lineare di $2t - e$ equazioni per le incognite $\sigma_1, \ldots, \sigma_e$ (infatti sappiamo già che $\sigma_0 = 1$). Una soluzione di questo sistema esiste: è la nostra $\sigma(Z)$. Mostriamo ora che tale soluzione è automaticamente quella di grado minimo.

Sia $\tilde{\sigma} = \sum_{i=0}^{e} \tilde{\sigma}_i Z^i$, con $\tilde{\sigma}_0 = 1$, la soluzione di grado minimo; per $e+1 \leqslant k \leqslant 2t$ abbiamo

$$0 = \sum_l S_{k-l} \tilde{\sigma}_l = \sum_{i \in M} \sum_l e_i \beta^{(k-l)i} \tilde{\sigma}_l = \sum_{i \in M} e_i \beta^{ik} \tilde{\sigma}(\beta^{-i}).$$

Possiamo interpretare il membro di destra di quest'equazione come un sistema lineare per $e_i\tilde{\sigma}(\beta^{-i})$ con coefficienti β^{ik}. La matrice dei coefficienti è di tipo Vandermonde, quindi il suo determinante è diverso da 0. Di conseguenza il sistema ha l'unica soluzione $e_i\tilde{\sigma}(\beta^{-i}) = 0$, ma $e_i \neq 0$, pertanto $\sigma(Z)$ divide $\tilde{\sigma}(Z)$, e $\sigma(Z) = \tilde{\sigma}(Z)$ per minimalità.

Come possiamo trovare la soluzione senza però dover risolvere il sistema lineare? Innanzitutto, detto $S(Z) := \sum S_l Z^l$ il *polinomio delle sindromi*, riscriviamo la condizione (4.1) come

$$S(Z)\sigma(Z) \equiv \omega(Z) \pmod{Z^{2t}}. \tag{4.2}$$

L'equazione (4.2) è chiamata *equazione chiave*. Un algoritmo per determinare $\sigma(Z)$ e $\omega(Z)$ è l'*algoritmo di Berlekamp-Massey*. Qui mostriamo un algoritmo simile, basato sull'algoritmo euclideo di calcolo dell'MCD.[1]

Supponiamo di voler trovare il massimo comune divisore tra i polinomi $f(Z)$ e $g(Z)$, con $\deg(f) \geqslant \deg(g)$. Chiamiamo $f^{(0)} := f$ e $g^{(0)} := g$. Il primo passo dell'algoritmo euclideo permette di scrivere

$$f^{(0)} = q^{(0)}g^{(0)} + g^{(1)}$$

con $\deg(g^{(1)}) < \deg(g^{(0)})$. Chiaramente l'MCD tra f e g divide $g^{(1)}$. Battezzando $f^{(1)} := g^{(0)}$ possiamo scrivere

$$\begin{pmatrix} f^{(1)} \\ g^{(1)} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q^{(0)} \end{pmatrix} \begin{pmatrix} f^{(0)} \\ g^{(0)} \end{pmatrix} = B^{(1)} \begin{pmatrix} f \\ g \end{pmatrix}$$

dove chiamiamo

$$G^{(0)} = \begin{pmatrix} 0 & 1 \\ 1 & -q^{(0)} \end{pmatrix}, \qquad B^{(0)} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

e ovviamente $B^{(1)} = G^{(0)}B^{(0)}$.

Se il resto non è zero, si può ripetere il procedimento con $f^{(1)}$ e $g^{(1)}$ ottenendo

$$f^{(1)} = q^{(1)}g^{(1)} + g^{(2)}$$

con $\deg(g^{(2)}) < \deg(g^{(1)})$. Come sopra, detto $f^{(2)} := g^{(1)}$, si ha

$$\begin{pmatrix} f^{(2)} \\ g^{(2)} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q^{(1)} \end{pmatrix} \begin{pmatrix} f^{(1)} \\ g^{(1)} \end{pmatrix} = G^{(1)}B^{(1)} \begin{pmatrix} f \\ g \end{pmatrix} = G^{(2)} \begin{pmatrix} f \\ g \end{pmatrix},$$

[1] Questo algoritmo *non* è stato presentato durante il corso. L'algoritmo visto a lezione è simile: si fissa un ordinamento su $(\sigma(Z), \omega(Z)) \in (\mathbb{F}_{q^r}[Z])^2$, e si effettua una divisione simile a quella tra polinomi multivariati, a partire dalla divisione tra le soluzioni banali $(1, S(Z))$ e $(0, Z^{2t})$.

con

$$G^{(1)} = \begin{pmatrix} 0 & 1 \\ 1 & -q^{(1)} \end{pmatrix}$$

e $B^{(2)} = G^{(1)}B^{(1)} = G^{(1)}G^{(0)}B^{(0)}$. Questo procedimento può essere ripetuto finché il resto è diverso da zero; il passo r-esimo in generale avrà

$$f^{(r)} = q^{(r)}g^{(r)} + g^{(r+1)},$$

$$B^{(r+1)} = \begin{pmatrix} 0 & 1 \\ 1 & -q^{(r)} \end{pmatrix} B^{(r)} = G^{(r)}B^{(r)} = G^{(r)}G^{(r-1)} \cdots G^{(0)},$$

$$\begin{pmatrix} f^{(r+1)} \\ g^{(r+1)} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q^{(r)} \end{pmatrix} \begin{pmatrix} f^{(r)} \\ g^{(r)} \end{pmatrix} = B^{(r+1)} \begin{pmatrix} f \\ g \end{pmatrix}.$$

Dato che il grado decresce ad ogni passo, a un certo punto (diciamo $R = r + 1$) si avrà $g^{(R)} = 0$; in particolare

$$\begin{pmatrix} f^{(R)} \\ 0 \end{pmatrix} = B^{(R)} \begin{pmatrix} f \\ g \end{pmatrix}. \tag{4.3}$$

Esplicitando

$$B^{(R)} = \begin{pmatrix} B_{11}^{(R)} & B_{12}^{(R)} \\ B_{21}^{(R)} & B_{22}^{(R)} \end{pmatrix},$$

la prima riga dell'equazione (4.3) recita

$$f^{(R)} = B_{11}^{(R)} f + B_{12}^{(R)} g \tag{4.4}$$

e quindi $\mathrm{MCD}(f, g)$ divide $f^{(R)}$.

La matrice $B^{(R)}$ è invertibile, poiché $\det B^{(R)} = \prod_{i=0}^{R-1} \det G^{(i)} = (-1)^R$; detta $D^{(R)} := \left(B^{(R)}\right)^{-1}$, moltiplicando per $D^{(R)}$ l'equazione (4.3) si ottiene

$$\begin{aligned} f &= D_{11}^{(R)} f^{(R)}, \\ g &= D_{21}^{(R)} f^{(R)}, \end{aligned} \tag{4.5}$$

di conseguenza $f^{(R)}$ divide $\mathrm{MCD}(f, g)$, e quindi è il massimo comun divisore (a meno di costanti moltiplicative).

Notiamo in particolare che le formule (4.5) ci permettono di ricavare esplicitamente f e g a partire da $B^{(R)}$ e $f^{(R)}$:

$$\begin{aligned} f &= (-1)^R B_{22}^{(R)} f^{(R)}, \\ g &= (-1)^{R+1} B_{21}^{(R)} f^{(R)}. \end{aligned}$$

Vogliamo a questo punto risolvere l'equazione (4.2). Nell'algoritmo appena presentato, al passo r-esimo si ha

$$\begin{aligned} g^{(r)} &= B_{21}^{(r)} f + B_{22}^{(r)} g \\ &= B_{22}^{(r)} g \pmod{f} \end{aligned}$$

che ha la stessa struttura dell'equazione chiave, ponendo $f(Z) = Z^{2t}$ e $g(Z) = S(Z)$. Mostreremo che si ha $\omega(Z) = g^{(r)}(Z)$ e $\sigma(Z) = B_{22}^{(r)}$ per qualche passo r della ricorsione, diciamo r'. È sufficiente trovare r' tale che $\deg(B_{22}^{(r')}) \leqslant t$ e $\deg(g^{(r')}) \leqslant t-1$. Dato che $g^{(r)}$ è il resto ad ogni passo, il suo grado parte da $\deg(S) = 2t$ per $r = 0$ e cala al crescere di r. Sia dunque r' tale che

$$\deg(g^{(r'-1)}) \geqslant t \text{ e } \deg(g^{(r')}) \leqslant t-1;$$

in questo modo siamo certi che $\deg(g^{(r')}) \leqslant t-1$. Abbiamo finito se dimostriamo che $\deg(B_{22}^{(r')}) \leqslant t$.

Innanzitutto,

$$B^{(r')} = \prod_{r=r'-1}^{0} \begin{pmatrix} 0 & 1 \\ 1 & -q^{(r)} \end{pmatrix}.$$

Poiché il grado dei resti successivi cala, il grado dei quozienti $q^{(r)}$ è almeno 1. Procedendo nelle moltiplicazioni, si può vedere che $\deg(B_{22}^{(r')}) > \deg(B_{12}^{(r')})$. Possiamo concludere che $\deg(B_{22}^{(r')})$ cresce al crescere di r, partendo da 1 per $r = 1$. Ricordiamo inoltre che $f^{(r')} = g^{(r'-1)}$, dunque $\deg(f^{(r')}) > \deg(g^{(r')})$. Dalle formule (4.5), si ricava facilmente

$$f = (-1)^{r'} \left(B_{22}^{(r')} f^{(r')} - B_{12}^{(r')} g^{(r')} \right).$$

Dalle disuguaglianze sui gradi appena trovate,

$$\deg(f) = \deg\left(B_{22}^{(r')} f^{(r')} \right) = \deg(B_{22}^{(r')}) + \deg(f^{(r')}),$$

e quindi

$$\deg(B_{22}^{(r')}) = \deg(f) - \deg(f^{(r')}) = \deg(Z^{2t}) - \deg(g^{(r'-1)}) \leqslant 2t - t = t$$

e questo prova la nostra tesi.

4.3 Codici Reed-Solomon

Definizione 4.4. Un *codice di Reed-Solomon* (o *codice RS*) è un codice BCH di lunghezza $n = q - 1$ su $\mathbb{F}_q$. Se ha distanza designata δ, il suo generatore è

$$g(X) = \prod_{i=\ell}^{\ell+\delta-2} (X - \beta^i)$$

dove β è un elemento primitivo di $\mathbb{F}_q$.

Proposizione 4.5. *Un codice RS di distanza designata δ ha effettivamente distanza $d = \delta$.*

Dimostrazione. Un codice RS ha dimensione $n-\delta+1$ e dunque dalla proposizione 2.6 abbiamo $d \leqslant n+1-n+\delta-1 = \delta$. Dato che l'altra disuguaglianza vale in generale per i codici BCH, abbiamo l'uguaglianza richiesta. □

4.4 Polinomio di Mattson-Solomon

Sia R un anello qualsiasi (commutativo con identità), e α una radice n-esima principale dell'unità, ovvero tale che

(1) $\alpha^n = 1$;

(2) $\sum_{j=0}^{n-1} \alpha^{jk} = 0 \quad \forall k = 1, \dots, n-1.$

Se R è un dominio, è sufficiente prendere una radice n-esima primitiva dell'unità, rimpiazzando (2) con

(2′) $\alpha^k \neq 1 \quad \forall k = 1, \dots, n-1.$

Per comodità, indicheremo le coordinate di $\boldsymbol{v} \in R^n$ come $(v_0, \dots, v_{n-1})$. La mappa

$$\begin{aligned} \mathcal{F} : R^n &\longrightarrow R^n \\ \boldsymbol{v} &\longmapsto \mathbf{f} \end{aligned}$$

con

$$f_k = \sum_{j=0}^{n-1} v_j \alpha^{jk}, \quad k = 0, \dots, n-1$$

è detta *trasformata discreta di Fourier*.

Vediamo cosa si può fare nel nostro caso. Ricordiamo che siamo nell'ambito dei codici ciclici, quindi stiamo lavorando in $\mathbb{K}[X]/(X^n-1)$, e in questo anello il polinomio X è invertibile:

$$X\,X^{n-1} = X^n = 1 \pmod{X^n - 1}.$$

Più in generale ha senso X^r per ogni $r \in \mathbb{Z}$, ed è equivalente a $X^{[r]}$, dove $[r] \in \{0, \dots, n-1\}$ indica il rappresentante di $r \pmod n$. Consideriamo allora il polinomio $v(X) = v_0 + v_1 X + \cdots + v_{n-1} X^{n-1}$.

Definizione 4.6. Siano α e $\nu(X)$ come sopra. Il *polinomio di Mattson-Solomon* di $\nu(X)$ è il polinomio

$$\hat{\nu}(X) = \sum_{j=0}^{n-1} \nu(\alpha^{-j}) X^j = \sum_{j=1}^{n} \nu(\alpha^j) X^{n-j}.$$

Osserviamo che, se $\boldsymbol{\nu}$ e $\nu(X)$ sono associati, cioè

$$\boldsymbol{\nu} = (\nu_0, \dots, \nu_{n-1}) \iff \nu(X) = \nu_0 + \nu_1 X + \cdots + \nu_{n-1} X^{n-1},$$

alla n-upla $\mathcal{F}(\boldsymbol{\nu})$ è associato il polinomio $\hat{\nu}(X^{-1})$, cioè il polinomio

$$\sum_{j=0}^{n-1} \hat{\nu}_{[-j]} X^j = \hat{\nu}_0 + \hat{\nu}_{n-1} X + \cdots + \hat{\nu}_1 X^{n-1}.$$

Alcune delle principali proprietà del polinomio di Mattson-Solomon (o equivalentemente della trasformata discreta di Fourier) sono racchiuse nelle seguenti proposizioni.

Proposizione 4.7. *Per* $a(X), b(X) \in \mathbb{K}[X]/(X^n - 1)$*, denotiamo con* ab *l'usuale prodotto tra polinomi e con* $a \cdot b$ *il prodotto coefficiente per coefficiente, ovvero*

$$(a \cdot b)(X) = \sum_{j=0}^{n-1} a_j b_j X^j.$$

Allora

$$\widehat{ab} = \hat{a} \cdot \hat{b}.$$

Dimostrazione. Ragionando coefficiente per coefficiente, si ha

$$(\widehat{ab})_k = (ab)(\alpha^{-k}) = \sum_{l=0}^{2(n-1)} \sum_{i=0}^{l} a_i b_{l-i} \alpha^{-kl} =$$
$$= \left(\sum_{i=0}^{n-1} a_i \alpha^{-ki} \right) \left(\sum_{i=0}^{n-1} b_i \alpha^{-ki} \right) = a(\alpha^{-k}) b(\alpha^{-k}) = (\hat{a})_k (\hat{b})_k$$

e questo prova la tesi. □

Proposizione 4.8 (Formula di inversione)**.** *Per* $a(X) \in \mathbb{K}[X]/(X^n - 1)$*, vale*

$$a(X) = \frac{1}{n} \hat{\hat{a}}(X^{-1})$$

dove $n = 1 + \cdots + 1$ *(*n *volte) in* $\mathbb{K}$ *(che è invertibile: ricordiamo che* n *è primo con* $\mathrm{char}(\mathbb{K})$*).*

Dimostrazione. Anche in questo caso, ragioniamo coefficiente per coefficiente.

$$\left(\hat{a}(X^{-1})\right)_k = \hat{a}(\alpha^k) = \sum_{j=0}^{n-1} a(\alpha^{-j})\alpha^{jk} =$$
$$= \sum_{j=0}^{n-1}\sum_{i=0}^{n-1} a_i \alpha^{-ij}\alpha^{jk} = \sum_{i=0}^{n-1} a_i \sum_{j=0}^{n-1} \alpha^{j(k-i)} = a_k n$$

dove nell'ultima uguaglianza si ha ovviamente $\sum \alpha^{j(k-i)} = 0$ se $k \neq i$, e $\sum \alpha^{j(k-i)} = n$ se $k = i$. □

Cosa c'entra tutto questo con i codici RS? In $\mathbb{K}[X]/(X^n-1)$ consideriamo il sottospazio dei polinomi di grado minore o uguale a $n-d$. Se $g = g_0 + g_1 X + \cdots + g_{n-d}X^{n-d}$, si ha che $\hat{g}(\alpha^{-k}) = n g_{n-k} = 0$ per $k = 1, \ldots, d-1$. In altre parole,

$$\{\hat{g}(X) \mid \deg(g) \leqslant n-d\} = \left\{ h \in \mathbb{K}[X]/(X^n-1) \;\middle|\; h(\alpha^{-1}) = \cdots = h(\alpha^{-(d-1)}) = 0 \right\}$$

che è un codice RS. Quindi il polinomio di Mattson-Solomon ci dà un modo veloce per codificare una parola di un codice RS, identificandola con un polinomio di grado al più $n-d$.

Capitolo 5

Codici di Goppa

I codici di Goppa sono l'ultima tecnica di codifica che presenteremo. Sono codici lineari che rappresentano una generalizzazione dei codici BCH.

Definizione 5.1. Sia $g(X)$ un polinomio monico di grado d su $\mathbb{F}_{q^r}$ e sia $L = \{\gamma_0, \ldots, \gamma_{n-1}\} \subseteq \mathbb{F}_{q^r}$ un insieme di punti tali che $g(\gamma_i) \neq 0 \quad \forall i = 0, \ldots, n-1$. Definiamo *codice di Goppa* su $\mathbb{F}_q$ associato al *supporto* L e al *polinomio di Goppa* g come l'insieme

$$\Gamma(L, g) := \left\{ (c_0, \ldots, c_{n-1}) \in (\mathbb{F}_q)^n \;\middle|\; \sum_{i=0}^{n-1} \frac{c_i}{X - \gamma_i} \equiv 0 \pmod{g(X)} \right\}.$$

Naturalmente i codici di Goppa sono lineari. Notiamo che $X - \gamma_i$ e $g(X)$ sono coprimi perché $g(\gamma_i) \neq 0$, quindi $X - \gamma_i$ è invertibile modulo $g(X)$. È possibile calcolare esplicitamente l'inverso:

$$(X - \gamma_i)^{-1} = -\frac{1}{g(\gamma_i)} \frac{g(X) - g(\gamma_i)}{X - \gamma_i} \pmod{g(X)}$$

in cui la divisione a secondo membro è una divisione esatta (se si vuole, per teorema di Ruffini).

Scegliendo $g(X) = X^{\delta-1}$, e $L = \{\alpha^{-i} \mid i = 0, \ldots, n-1\}$, con α radice primitiva n-esima dell'unità in $\mathbb{F}_{q^r}$ otteniamo un codice BCH di distanza designata δ, generato da $\prod_{i=1}^{\delta-1}(X - \alpha^i)$. Infatti, per $i = 0, \ldots, n-1$,

$$(X - \alpha^{-i})^{-1} = -\sum_{j=0}^{\delta-2} \alpha^{i(j+1)} X^j \pmod{X^{\delta-1}},$$

quindi per una parola $\mathbf{c} = (c_0, \dots, c_{n-1})$ la condizione di appartenenza al codice $\Gamma(L, g)$ diventa

$$\sum_{i=0}^{n-1} \sum_{j=0}^{\delta-2} c_i \alpha^{i(j+1)} X^j = 0$$

cioè, detto $c(X) = c_0 + c_1 X + \cdots + c_{n-1} X^{n-1}$,

$$\sum_{j=0}^{\delta-2} c(\alpha^{j+1}) X^j = 0$$

da cui $c(\alpha^{j+1}) = 0$ per ogni $j = 0, \dots, \delta - 2$.

5.1 Matrice di parità e decodifica

Vogliamo adesso costruire la matrice di parità per un codice di Goppa. Sappiamo che se $\mathbf{c} \in \Gamma(L, g)$, allora per definizione $\mathbf{c} \cdot M = 0$, dove

$$M = \begin{pmatrix} (X-\gamma_0)^{-1} \\ (X-\gamma_1)^{-1} \\ \vdots \\ (X-\gamma_{n-1})^{-1} \end{pmatrix} \pmod{g(X)}.$$

Cerchiamo di riscrivere meglio questa matrice. Siano $h_j := g(\gamma_j)^{-1}$; ricavando esplicitamente i polinomi $(X-\gamma_j)^{-1}$ e sostituendoli con la riga dei loro coefficienti si ottiene

$$\begin{pmatrix} h_0 g_d & h_0(g_{d-1} + g_d\gamma_0) & \cdots & h_0(g_1 + g_2\gamma_0 + \cdots + g_d\gamma_0^{d-1}) \\ h_1 g_d & h_1(g_{d-1} + g_d\gamma_1) & \cdots & h_1(g_1 + g_2\gamma_1 + \cdots + g_d\gamma_1^{d-1}) \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-1} g_d & h_{n-1}(g_{d-1} + g_d\gamma_{n-1}) & \cdots & h_{n-1}(g_1 + g_2\gamma_{n-1} + \cdots + g_d\gamma_{n-1}^{d-1}) \end{pmatrix}$$

che può essere riscritta come

$$\begin{pmatrix} h_0 & h_0\gamma_0 & \cdots & h_0\gamma_0^{d-1} \\ h_1 & h_1\gamma_1 & \cdots & h_1\gamma_1^{d-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-1} & h_{n-1}\gamma_{n-1} & \cdots & h_{n-1}\gamma_{n-1}^{d-1} \end{pmatrix} \begin{pmatrix} g_d & g_{d-1} & \cdots & g_1 \\ 0 & g_d & \cdots & g_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & g_d \end{pmatrix}.$$

La matrice a destra è invertibile, e può essere considerata una matrice di cambiamento di coordinate. È quindi possibile usare come matrice di parità la matrice

di sinistra, che denoteremo con H. In effetti, tale matrice sta in $\mathcal{M}_{n\times d}(\mathbb{F}_{q^r})$, e può essere ricondotta a una matrice di $\mathcal{M}_{n\times rd}(\mathbb{F}_q)$ esprimendo nel solito modo un elemento di $\mathbb{F}_{q^r}$ con r cifre di $\mathbb{F}_q$.

Proposizione 5.2. *La dimensione di un codice di Goppa è almeno* $n - rd$*, e la sua distanza è almeno* $d + 1$*.*

Dimostrazione. Il rango della matrice H è al massimo rd (non è detto che le colonne siano indipendenti). Dunque $n - \dim(\Gamma(L, g)) \leqslant rd$, da cui la prima parte della tesi. Per la seconda parte, è sufficiente vedere che scegliendo d righe di H, esse risultano linearmente indipendenti. Un rapido conto mostra che un minore $d \times d$ è di tipo Vandermonde, quindi invertibile; la tesi è così dimostrata. □

La decodifica dei codici di Goppa procede in modo simile ai codici BCH. Chiamiamo $\boldsymbol{v}$ la parola inviata, $\boldsymbol{w}$ la parola ricevuta ed $\boldsymbol{e} = \boldsymbol{w} - \boldsymbol{v}$ l'errore. Siano

$$M := \{i = 0, \dots, n-1 \mid e_i \neq 0\}$$

l'insieme delle posizioni degli errori, e

$$s(X) := \sum_{i=0}^{n-1} \frac{e_i}{X - \gamma_i} \pmod{g(X)}$$

la sindrome polinomiale. Notiamo che il ricevente è in grado di calcolare $s(X)$ a partire da $\boldsymbol{w}$ e $g(X)$:

$$s(X) = \sum_{i=0}^{n-1} \frac{e_i}{X - \gamma_i} = \sum_{i=0}^{n-1} \frac{w_i - v_i}{X - \gamma_i} = \sum_{i=0}^{n-1} \frac{w_i}{X - \gamma_i} \pmod{g(X)}.$$

Definiamo i polinomi locatore e valutatore d'errore:

$$\sigma(X) = \prod_{i\in M} (X - \gamma_i),$$

$$\omega(X) = \sum_{i\in M} e_i \prod_{j\in M\setminus\{i\}} (X - \gamma_j).$$

Con queste definizioni, è facile verificare che vale l'*equazione chiave per codici di Goppa*:

$$s(X)\sigma(X) \equiv \omega(X) \pmod{g(X)}.$$

Per trovare $\sigma(X)$ e $\omega(X)$, e dunque anche posizioni e valori degli errori, si procede come nella sezione 4.2.

Parte II

Crittografia

Capitolo 6

Introduzione

Supponiamo che due persone, tradizionalmente chiamate Alice e Bob, vogliano scambiarsi delle informazioni segrete. Possono seguire due strade:

- *nascondere* il messaggio in un qualche mezzo apparentemente privo di informazioni rilevanti, in modo che un intercettatore non si accorga della presenza del messaggio;
- *modificare* il messaggio, in modo da rendere impossibile recuperarlo a meno di conoscere informazioni aggiuntive.

La prima strada è percorsa dalla *steganografia*, la seconda dalla *crittografia*. Il nostro lavoro si concentrerà proprio su quest'ultima.

I protocolli crittografici possono essere applicati in diverse situazioni:

Cifratura. Alice vuole inviare un messaggio a Bob, evitando che un intercettatore (tradizionalmente, Eva) ne conosca il contenuto.

Autenticazione. Alice vuole provare a Bob la propria identità.

Firma. Alice vuole che tutti e soli i propri documenti siano riconosciuti come tali.

Supporremo sempre che il canale attraverso cui viaggiano le informazioni sia *pubblico* e quindi potenzialmente accessibile a tutti. Supporremo inoltre che il protocollo di cifratura sia pubblico.

6.1 Concetti di base

Vediamo qui quali sono i principali ingredienti per costruire un crittosistema:

- un insieme $\mathcal{M}$ (*spazio dei messaggi in chiaro*), formato da stringhe di simboli di un dato alfabeto;
- un insieme $\mathcal{C}$ (*spazio dei messaggi cifrati*), formato da stringhe di simboli di un dato alfabeto, anche diverso dall'alfabeto di $\mathcal{M}$;
- un insieme $\mathcal{K}$ (*spazio delle chiavi*), i cui elementi sono detti (per l'appunto) *chiavi*;
- per ogni $e \in \mathcal{K}$, una mappa $E_e : \mathcal{M} \longrightarrow \mathcal{C}$ biiettiva[1] (*funzione di cifratura*);
- per ogni $d \in \mathcal{K}$, una mappa $D_d : \mathcal{C} \longrightarrow \mathcal{M}$ biiettiva (*funzione di decifratura*).

Definizione 6.1. Con le notazioni introdotte sopra, un *sistema di cifratura* (o *cifrario*) è formato da un insieme di funzioni di cifratura $\mathcal{E} = \{E_e \mid e \in \mathcal{K}\}$ e un corrispondente insieme di funzioni di decifratura $\mathcal{D} = \{D_d \mid d \in \mathcal{K}\}$ tali che per ogni $e \in \mathcal{K}$ esiste un'unica $d \in \mathcal{K}$ tale che $D_d(E_e(m)) = m$ per ogni $m \in \mathcal{M}$.

Un crittosistema, quindi, è una quintupla

$$(\mathcal{M}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D}).$$

Naturalmente è nel nostro interesse fare in modo che il cifrario sia sicuro, cioè che un eventuale terzo che voglia decodificare il nostro messaggio, sostituirsi a noi o firmare documenti a nostro nome trovi grosse difficoltà nel farlo, ossia che gli richieda molto tempo o abbia basse probabilità di riuscita. Dovremo dunque tenere conto del *costo computazionale* delle operazioni che useremo nei nostri algoritmi.

Operazione	**Algoritmo**	**Costo**
Somma	Algoritmo usuale	$\mathcal{O}(\log N)$
Prodotto	Algoritmo usuale	$\mathcal{O}((\log N)^2)$
Potenza a^b	Ricorsivo: $a^b = (a^{\lfloor b/2 \rfloor})^2 \; a^{b \bmod 2}$	$\mathcal{O}(\log b (\log a)^2)$
Inverso in $\mathbb{F}_q$	Elevazione alla $q-2$	$\mathcal{O}((\log q)^3)$

Tabella 6.1: Costo computazionale di alcune operazioni comuni. Qui N indica il numero di cifre degli operandi.

[1] In realtà è sufficiente iniettiva, in modo che sia biiettiva sull'immagine.

6.2 Problemi computazionalmente difficili

Alcuni tipi di funzioni sono più adatti di altri per rappresentare funzioni di cifratura. Le definizioni seguenti saranno lasciate volutamente poco rigorose e intuitive.

Definizione 6.2. Dati due insiemi X e Y, una funzione $f : X \longrightarrow Y$ è detta *one-way* se è "facile" calcolare $f(x)$ per ogni $x \in X$, mentre è "computazionalmente impossibile" per quasi ogni $y \in f(X)$ trovare un $x \in X$ tale che $f(x) = y$.

Definizione 6.3. Una funzione *one-way* $f : X \longrightarrow Y$ è detta *trapdoor*[2] se diventa possibile trovare $x \in X$ tale che $f(x) = y$ per ogni $y \in f(X)$, note alcune informazioni extra (*trapdoor information*).

L'esistenza di funzioni *one-way* o *trapdoor one-way*, anche dopo averne dato delle definizioni precise, non è stata dimostrata; alcune funzioni sono candidate *one-way*, ma non si sa se in futuro si possano trovare algoritmi più efficienti per invertirle.

Per costruire funzioni *one-way*, l'idea principale è quella di studiare alcuni problemi *computazionalmente difficili*. Noi ne analizzeremo principalmente due.

Fattorizzazione. Dato $n \in \mathbb{Z}$, $n = pq$, determinare p e q.

Logaritmo discreto. Dato G gruppo ciclico finito generato da g, e $a \in G$, trovare $b \in \mathbb{Z}$ tale che $g^b = a$.

[2]Qui *trapdoor* è un attributo di *one-way*.

Capitolo 7

Test di primalità

Numerosi sistemi crittografici richiedono l'uso di numeri primi. Sorge dunque il problema di stabilire, dato un certo numero n, se esso è primo oppure no: questo compito è svolto dai *test di primalità*.

I test di primalità più efficienti noti (con alcune significative eccezioni) sono *probabilistici*: se rispondono `NO`, sicuramente il numero è composto, mentre se rispondono `SÌ` non è detto che il numero sia primo. Tali test, comunque, forniscono anche una stima della probabilità di errore.

7.1 Il crivello di Eratostene

Il crivello di Eratostene è il metodo più semplice per ottenere tabelle di numeri primi. L'idea è: si scrivono tutti i numeri da 2 a n, dopodiché si prende il più piccolo numero non cancellato (all'inizio è il 2), lo si segna come primo e si cancellano tutti i suoi multipli.

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~

Figura 7.1: Risultato del crivello di Eratostene, per $n = 50$.

Si prosegue finché non si giunge a $\sqrt{n}$; i numeri rimasti dopo le cancellazioni sono primi.

Verificare se tutti i numeri da 2 a n siano divisori diventa inefficiente al crescere delle dimensioni dei numeri in gioco; in crittografia si utilizzano tipicamente grandi numeri primi (dell'ordine delle 300 cifre decimali).

7.2 Test di Fermat

Il test di Fermat si basa sul ben noto piccolo teorema di Fermat, di cui non forniamo una dimostrazione.

Teorema 7.1 (Piccolo teorema di Fermat)**.** *Sia n primo. Allora $a^{n-1} \equiv 1 \pmod{n}$ per ogni $a \in (\mathbb{Z}_n)^*$.*

Dunque, per vedere se un numero n è primo, scegliamo un $a < n$ e calcoliamo $a^{n-1} \pmod{n}$. Se non otteniamo 1, sicuramente n è composto.

Definizione 7.2. Sia n intero dispari, e a primo con n tale che $a^{n-1} \equiv 1 \pmod{n}$. In tal caso, n è detto *pseudoprimo* in base a.

Aumentando le basi a su cui testare la formula di Fermat, aumenta la probabilità che n sia primo; tuttavia, esistono dei numeri composti, detti *numeri di Carmichael*, che sono pseudoprimi in base a per ogni a primo con n. Questi numeri passano in ogni caso il test di Fermat pur essendo composti.

7.3 Test di Miller-Rabin

Uno dei test di primalità più utilizzati è il *test di Miller-Rabin*. Questo test si basa su un semplice risultato.

Proposizione 7.3. *Sia n un primo dispari e sia $n - 1 = 2^s r$ dove r è dispari. Sia a un numero primo con n. Allora si ha, in alternativa:*

1. $a^r \equiv 1 \pmod{n}$;
2. $a^{2^j r} \equiv -1 \pmod{n}$ *per qualche* $j = 0, \ldots, s-1$.

Dimostrazione. Chiaramente, se n è primo, $\mathbb{Z}_n$ è un campo, e quindi il polinomio $X^2 - 1$ ha solo 1 e -1 come radici. Dal piccolo teorema di Fermat

$$a^{2^s r} \equiv 1 \pmod{n}$$

quindi $a^{2^{s-1} r} \equiv \pm 1 \pmod{n}$. Se otteniamo -1, vale (2) con $j = s - 1$; altrimenti, $a^{2^{s-1} r} \equiv 1 \pmod{n}$, e quindi $a^{2^{s-2} r} \equiv \pm 1 \pmod{n}$. Continuando in questo modo, se a un certo punto si ottiene -1, abbiamo la tesi (2); altrimenti, si giunge a $a^r \equiv 1 \pmod{n}$ che è la tesi (1). □

Consideriamo dunque la contronominale: se riusciamo a trovare a primo con n tale che $a^r \not\equiv 1 \pmod{n}$ e $a^{2^j r} \not\equiv -1 \pmod{n}$ per ogni $j = 0, \ldots, s-1$, sicuramente n è composto.

Algoritmo MILLER-RABIN

Input: n candidato primo

1: $n - 1 = 2^s r$
2: Scelta di $a \in \{2, \ldots, n-2\}$
3: $x := a^r \pmod{n}$
4: **if** $x \equiv 1 \pmod{n}$ **then**
5: **return** `PROBABILE PRIMO`
6: **else**
7: **for** $j := 0$ **to** $s - 1$ **do**
8: **if** $x \equiv -1 \pmod{n}$ **then**
9: **return** `PROBABILE PRIMO`
10: **else**
11: $x := x^2 \pmod{n}$
12: **end if**
13: **end for**
14: **return** `COMPOSTO`
15: **end if**

Output: `COMPOSTO` oppure `PROBABILE PRIMO`

Se l'algoritmo MILLER-RABIN risponde `COMPOSTO`, sicuramente n è composto. Si può dimostrare che la probabilità che MILLER-RABIN risponda `PROBABILE PRIMO` con n composto è minore di $1/4$. Ripetendo dunque il test per m valori diversi di a, quindi, la probabilità di ottenere sempre un falso positivo è di $(1/4)^m$.

7.4 Test di Pocklington

Teorema 7.4 (Pocklington)**.** *Sia n un intero. Supponiamo che esista q primo tale che $q \mid n-1$ e $q > \sqrt{n} - 1$. Se esiste a tale che*

1. $a^{n-1} \equiv 1 \pmod{n}$,
2. $\mathrm{MCD}(a^{\frac{n-1}{q}} - 1, n) = 1$,

allora n è primo.

Dimostrazione. Supponiamo che n non sia primo. Allora esiste p primo tale che $p \mid n$ e $p \leqslant \sqrt{n}$. Dato che $q > p-1$, si ha $\mathrm{MCD}(q, p-1) = 1$, quindi esiste u tale che $uq \equiv 1 \pmod{p-1}$. Ovviamente la congruenza modulo n implica quella modulo p; si ha, allora,

$$a^{\frac{n-1}{q}} \equiv a^{uq\frac{n-1}{q}} \equiv a^{u(n-1)} \equiv 1 \pmod{p},$$

dove si è usato il piccolo teorema di Fermat nella prima congruenza. Quindi $p \mid a^{\frac{n-1}{q}} - 1$, in contraddizione con l'ipotesi 2. □

Per verificare la primalità per n, allora, basta trovare q e a che soddisfino le ipotesi del teorema. Normalmente a non è un problema (funziona abbastanza spesso $a = 2$). Tuttavia l'esistenza di q non è affatto garantita: per esempio, per $n \leqslant 10000$, i primi per cui esiste un tale q sono circa il $57,8\%$ del totale. È possibile migliorare questo test per aumentare i primi che lo passano.

Corollario 7.5. *Sia n un intero. Supponiamo che $n-1$ si possa scrivere come prodotto di due numeri coprimi a, b con $a > \sqrt{n}$. Se per ogni fattore primo q di a esiste un intero a_q tale che $a_q^{n-1} \equiv 1 \pmod{n}$ e $\mathrm{MCD}\left(a_q^{\frac{n-1}{q}} - 1, n\right) = 1$, allora n è primo. Vale anche il viceversa.*

Dimostrazione. Mostreremo che un eventuale fattore primo p di n dev'essere maggiore di $\sqrt{n}$.

Sia p un primo che divide n. Scriviamo $a = q^e t$, con $q \nmid t$. Poniamo $c_q := a_q^{\frac{n-1}{q^e}} \pmod{p}$. Dunque ${c_q}^{q^e} \equiv a_q^{n-1} \equiv 1 \pmod{p}$. Ora, ${c_q}^{q^{e-1}} \not\equiv 1 \pmod{p}$ perché altrimenti si avrebbe $1 \equiv {c_q}^{q^{e-1}} \equiv a_q^{\frac{n-1}{q}} \pmod{p}$, in contraddizione con $\mathrm{MCD}\left(a_q^{\frac{n-1}{q}} - 1, n\right) = 1$. Quindi c_q ha ordine q^e in $(\mathbb{Z}_p)^*$, da cui si ricava che $q^e \mid p-1$. Ripetendo il ragionamento per ogni fattore q di a si ottiene che $a \mid p-1$, da cui $p > a > \sqrt{n}$. Quindi tutti i fattori primi di n sono maggiori di $\sqrt{n}$, che è assurdo.

Per il viceversa basta scegliere come a_q un generatore moltiplicativo di $\mathbb{Z}_q$. □

Il criterio di Pocklington funziona quindi fattorizzando parzialmente $n-1$ e cercando gli a_q che soddisfino le ipotesi del teorema. Naturalmente il criterio è ricorsivo: è possibile utilizzarlo per verificare la primalità di un candidato fattore q di $n-1$.

La grossa difficoltà nell'utilizzo di questo algoritmo consiste sostanzialmente nel fattorizzare $n-1$; il calcolo degli a_q è relativamente facile.

Capitolo 8

Fattorizzare su $\mathbb{Z}$

Come abbiamo detto, un problema studiato al fine di ricavare protocolli crittografici è la fattorizzazione. Presentiamo qui alcuni algoritmi utilizzati a tal scopo.

8.1 Residui quadratici

Definizione 8.1. Sia $p > 2$ un primo. Un intero a tale che $p \nmid a$ è detto *residuo quadratico modulo* p se esiste x tale che $x^2 \equiv a \pmod{p}$.

Perché introduciamo i residui quadratici in un capitolo sulla fattorizzazione?

Teorema 8.2. *Sia n prodotto di due primi. La conoscenza delle radici quadrate di un residuo quadratico a modulo n è equivalente alla conoscenza della fattorizzazione di n.*

Dimostrazione. $\boxed{\Leftarrow}$ Supponiamo di conoscere $n = pq$, e sia $a \in \mathbb{Z}_n$. Per il teorema cinese del resto, $\mathbb{Z}_n \simeq \mathbb{Z}_p \times \mathbb{Z}_q$: è sufficiente dunque trovare le radici di a modulo un primo.

Osserviamo, tra l'altro, che questo isomorfismo ci dice che un residuo quadratico $a \in \mathbb{Z}_n$ ha al massimo quattro radici quadrate:

- se $a \equiv 0 \pmod{n}$, ha un'unica radice quadrata (che è 0);
- se $a \in (\mathbb{Z}_n)^*$, per teorema cinese del resto $[a]_p \in (\mathbb{Z}_p)^*$ e $[a]_q \in (\mathbb{Z}_q)^*$, e ciascuno di questi ha due radici quadrate ($\mathbb{Z}_p$ e $\mathbb{Z}_q$ sono campi), quindi a ha un totale di quattro radici quadrate in $\mathbb{Z}_n$;
- se $\mathrm{MCD}(a, n) = p$, si ha che $[a]_p \in (\mathbb{Z}_p)^*$ ma $[a]_q = 0$, quindi a ha solo due radici quadrate in $\mathbb{Z}_n$; completamente analogo il caso $\mathrm{MCD}(a, n) = q$.

Dunque, vogliamo trovare le radici quadrate di un residuo quadratico a in $\mathbb{Z}_p$ con p primo. Distinguiamo due casi.

- **$p \equiv 3 \pmod 4$.** Innanzitutto -1 non è un quadrato modulo p. Infatti, se $a^2 \equiv -1 \pmod p$, si avrebbe $a^4 \equiv 1 \pmod p$, e quindi l'ordine di a in $(\mathbb{Z}_p)^*$ sarebbe 4. Ma 4 non divide l'ordine di $(\mathbb{Z}_p)^*$, che è $p-1$.

  Detto questo, è facile mostrare che, se a è un residuo quadratico modulo p, le sue radici quadrate sono $\pm a^{\frac{p+1}{4}}$:

  $$\left(\pm a^{\frac{p+1}{4}}\right)^4 \equiv a^{p+1} \equiv a^2 \pmod p$$

  da cui $\left(a^{\frac{p+1}{4}}\right)^2 \equiv \pm a \pmod p$. Tuttavia $-a$ viene escluso perché -1 non è un quadrato modulo p.

- **$p \equiv 1 \pmod 4$.** Andiamo a cercare una radice quadrata di a in un'estensione $\mathbb{F}_{p^2}$ di $\mathbb{Z}_p$. Consideriamo $b \in \mathbb{Z}_p$ in modo che $X^2 + bX + a \in \mathbb{Z}_p[X]$ sia irriducibile, e scriviamo

  $$\mathbb{F}_{p^2} \simeq \mathbb{Z}_p[X]/(X^2 + bX + a)\,.$$

  Indichiamo con $\alpha \in \mathbb{F}_{p^2}$ una radice di $X^2 + bX + a$. L'altra radice è α^p, quindi $X^2 + bX + a = (X-\alpha)(X-\alpha^p)$, e $\alpha^{p+1} = a$, quindi $\alpha^{\frac{p+1}{2}}$ è una radice quadrata di a in $\mathbb{F}_{p^2}$. Se tale radice sta in realtà nel sottocampo isomorfo a $\mathbb{Z}_p$, ho finito, altrimenti cambio polinomio irriducibile.

Quindi sappiamo come ricavare le radici quadrate modulo p primo, e quindi modulo n via teorema cinese del resto.
$\boxed{\Rightarrow}$ Supponiamo di conoscere le radici quadrate modulo n di un certo $a \in \mathbb{Z}_n$. Siano a_1, a_2, a_3, a_4 tali radici. Essendo le radici opposte a due a due, senza perdita di generalità supponiamo $a_1 = -a_2$ e $a_3 = -a_4$.

Sappiamo che n è composto, quindi vale l'isomorfismo dato dal teorema cinese del resto $\mathbb{Z}_n \simeq \mathbb{Z}_p \times \mathbb{Z}_q$, anche senza conoscere esplicitamente p e q. Abbiamo $(a_1 + a_3)(a_1 - a_3) = a_1^2 - a_3^2 \equiv 0 \pmod n$, e questo vale anche modulo p e q. Dunque $a_1 + a_3 \equiv 0 \pmod p$ oppure $a_1 - a_3 \equiv 0 \pmod p$ (e analogamente modulo q). Supponiamo per esempio $a_1 - a_3 \equiv 0 \pmod p$; allora si può ottenere p semplicemente calcolando $\mathrm{MCD}(a_1 - a_3, n) = p$. Tutti gli altri casi sono analoghi. □

Lavorando con residui quadratici, è utile introdurre una notazione.

Definizione 8.3. Sia p un primo dispari. Per un intero a, definiamo il *simbolo di Legendre*

$$\left(\frac{a}{p}\right) := \begin{cases} 0 & \text{se } p \mid a \\ 1 & \text{se } a \text{ è un residuo quadratico modulo } p \\ -1 & \text{se } a \text{ non è un residuo quadratico modulo } p. \end{cases}$$

Ovviamente, se $a \equiv b \pmod{p}$, si ha $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$. Vediamo alcune delle proprietà del simbolo di Legendre.

Proposizione 8.4 (Criterio di Eulero)**.** *Si ha che*

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p}.$$

Dimostrazione. Se $p \mid a$, entrambi i lati dell'equazione sono 0. Supponiamo che $p \nmid a$; dal piccolo teorema di Fermat $a^{p-1} \equiv 1 \pmod{p}$, quindi $a^{\frac{p-1}{2}} \equiv \pm 1 \pmod{p}$.

Sia g generatore di $(\mathbb{Z}_p)^*$, e sia $a = g^j$. Non è difficile mostrare che a è un residuo se e solo se j è pari. D'altra parte, $a^{\frac{p-1}{2}} = g^{j\frac{p-1}{2}}$ è 1 se e solo se $j\frac{p-1}{2}$ è divisibile per $p-1$, e quindi se e solo se j è pari. Quindi entrambi i membri dell'equazione assumono gli stessi valori per ogni $a \in \mathbb{Z}_p$. □

Corollario 8.5. *Per a, b interi e $p > 2$ primo vale*

$$\left(\frac{a}{p}\right)\left(\frac{b}{p}\right) = \left(\frac{ab}{p}\right).$$

Se b è primo con p, vale anche

$$\left(\frac{ab^2}{p}\right) = \left(\frac{a}{p}\right).$$

Dimostrazione. Dal criterio di Eulero:

$$\left(\frac{a}{p}\right)\left(\frac{b}{p}\right) \equiv a^{\frac{p-1}{2}} b^{\frac{p-1}{2}} \equiv (ab)^{\frac{p-1}{2}} \equiv \left(\frac{ab}{p}\right) \pmod{p}.$$

Naturalmente le equivalenze modulo p diventano uguaglianze perché 1, -1 e 0 sono diversi modulo p per ogni p.

La seconda uguaglianza è una diretta conseguenza della prima. □

Proposizione 8.6. *Per ogni primo $p > 2$ vale*

$$\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}.$$

Dimostrazione. Definiamo, per $n \in \mathbb{Z}$,

$$f(n) := \begin{cases} (-1)^{\frac{n^2-1}{8}} & \text{se } n \text{ è dispari} \\ 0 & \text{se } n \text{ è pari.} \end{cases}$$

È facile vedere che $f(nm) = f(n)f(m)$. Ora, $p^2 \equiv 1 \pmod 8$ per ogni p primo dispari, quindi in $\mathbb{F}_{p^2}$ esiste una radice ottava primitiva dell'unità, che indicheremo con ξ.

Sia ora $G := \sum_{j=0}^{7} f(j)\xi^j$. Si ha, sviluppando il conto,

$$\begin{aligned} G &= \xi - \xi^3 - \xi^5 + \xi^7 = \\ &= \xi - \xi^3 + \xi - \xi^3 = \\ &= 2(\xi - \xi^3) \end{aligned}$$

e quindi $G^2 = 4(\xi^2 - 2\xi^4 + \xi^6) = 8 \in \mathbb{Z}_p$. Notiamo che dunque $G \neq 0$ (perché $G^2 \neq 0$). Ora, da una parte di ha

$$G^p = (G^2)^{\frac{p-1}{2}} G = \left(\frac{8}{p}\right) G = \left(\frac{2}{p}\right) G,$$

e dall'altra

$$\begin{aligned} G^p &\overset{(1)}{=} \sum_{j=0}^{7} f(j)^p \xi^{jp} \overset{(2)}{=} \sum_{j=0}^{7} f(j)\xi^{jp} = \\ &\overset{(2)}{=} \sum_{j=0}^{7} f(p^2 j)\xi^{jp} \overset{(3)}{=} f(p) \sum_{h=0}^{7} f(h)\xi^h = \\ &= (-1)^{\frac{p-1}{8}} G, \end{aligned}$$

dove

(1) siamo in caratteristica p;

(2) discendono dalle proprietà di f;

(3) cambiamo variabile $h = jp$ (stiamo comunque sommando su tutte le classi modulo 8).

Da cui, semplificando $G \neq 0$, la tesi. □

Proposizione 8.7 (Legge di reciprocità quadratica)**.** *Siano p, q primi dispari. Allora*

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}.$$

Dimostrazione. Cerchiamo una radice primitiva q-esima dell'unità in un'estensione di $\mathbb{F}_p$. Per esempio, sia $\mathbb{K}$ il campo di spezzamento di $X^q - 1 \in \mathbb{F}_p[X]$, e sia $\xi \in \mathbb{K}$ di ordine q. Come nella proposizione precedente, definiamo

$$G := \sum_{j=0}^{q-1} \left(\frac{j}{q}\right) \xi^j.$$

Dimostreremo tra poco che $G^2 = (-1)^{\frac{q-1}{2}} q$ in $\mathbb{K}$. Supponendolo vero, scriviamo

$$G^p = (G^2)^{\frac{p-1}{2}} G = (-1)^{\frac{p-1}{2}\frac{q-1}{2}} q^{\frac{p-1}{2}} G = (-1)^{\frac{p-1}{2}\frac{q-1}{2}} \left(\frac{q}{p}\right) G,$$

e d'altra parte

$$G^p = \sum_{j=0}^{q-1} \left(\frac{j}{q}\right)^p \xi^{jp} = \sum_{j=0}^{q-1} \left(\frac{j}{q}\right) \xi^{jp} = \left(\frac{p}{q}\right) \sum_{j=0}^{q-1} \left(\frac{jp}{q}\right) \xi^{jp} = \left(\frac{p}{q}\right) G$$

da cui la tesi, semplificando $G \neq 0$. Manca solo da dimostrare che $G^2 = (-1)^{\frac{q-1}{2}} q$. Intanto scriviamo

$$\begin{aligned} G^2 &= \left(\sum_{j=0}^{q-1} \left(\frac{j}{q}\right) \xi^j\right) \left(\sum_{-k=0}^{q-1} \left(\frac{-k}{q}\right) \xi^{-k}\right) = \\ &= \left(\sum_{j=1}^{q-1} \left(\frac{j}{q}\right) \xi^j\right) \left(\sum_{-k=1}^{q-1} \left(\frac{-k}{q}\right) \xi^{-k}\right) = (\star) \end{aligned}$$

indicizzando la seconda somma su $-k$, e partendo da 1 (tanto per $j = 0$ e $k = 0$, i simboli di Legendre sono nulli). Raccogliamo $\left(\frac{-1}{q}\right)$ e cambiamo $-k$ in k:

$$\begin{aligned} (\star) &= \left(\frac{-1}{q}\right) \left(\sum_{j=1}^{q-1} \left(\frac{j}{q}\right) \xi^j\right) \left(\sum_{-k=1}^{q-1} \left(\frac{k}{q}\right) \xi^{-k}\right) = \\ &= \left(\frac{-1}{q}\right) \left(\sum_{j=1}^{q-1} \left(\frac{j}{q}\right) \xi^j\right) \left(\sum_{k=1}^{q-1} \left(\frac{k}{q}\right) \xi^{-k}\right) = \\ &= \left(\frac{-1}{q}\right) \sum_{j=1}^{q-1} \sum_{k=1}^{q-1} \left(\frac{jk}{q}\right) \xi^{j-k} = (\star\star). \end{aligned}$$

A questo punto sostituiamo k con jk (i termini scorrono su tutte le classi di equivalenza modulo q) e rimettiamo lo 0:

$$
\begin{aligned}
(\star\star) &= \left(\frac{-1}{q}\right) \sum_{j=1}^{q-1} \sum_{jk=1}^{q-1} \left(\frac{j^2 k}{q}\right) \xi^{j(1-k)} = \\
&= \left(\frac{-1}{q}\right) \sum_{k=0}^{q-1} \sum_{j=0}^{q-1} \left(\frac{k}{q}\right) \xi^{j(1-k)} = \\
&\overset{(\#)}{=} \left(\frac{-1}{q}\right) \sum_{j=0}^{q-1} \left(\frac{1}{q}\right) = \left(\frac{-1}{q}\right) q = (-1)^{\frac{q-1}{2}} q
\end{aligned}
$$

dove il passaggio (#) è dovuto al fatto che gli addendi della somma su k sono quasi tutti nulli, perché la somma su j è estesa a tutte le radici dell'unità; si salva solamente l'addendo con $k = 1$. Questo conclude la dimostrazione. □

8.2 Algoritmo ρ di Pollard

Supponiamo di sapere che un certo intero n sia composto. L'algoritmo ρ di Pollard è efficiente nel cercare fattori piccoli di n.

Sia $f : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n$ una qualsiasi funzione. (Tra le più usate, c'è $f(x) = x^2 + 1 \pmod{n}$.) Costruiamo una sequenza $(a_m) \subseteq \mathbb{Z}_n$ scegliendo a caso a_1 e definendo $a_{j+1} = f(a_j)$. Poiché i valori che può assumere f sono finiti, ad un certo punto la sequenza entrerà in un ciclo.

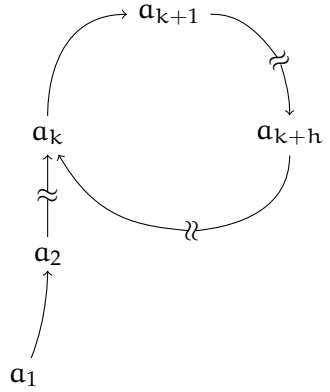


Figura 8.1: Diagramma della sequenza (a_m). La forma di questo diagramma ha dato il nome all'algoritmo.

Un'analisi probabilistica mostra che, per n grande, la lunghezza della coda della ρ è $\sqrt{(n\pi)/8}$, così come la lunghezza del ciclo. Il problema su cui si basa l'algoritmo è trovare una *collisione*, cioè due indici h, k tali che $a_h = a_k$. In media, questo accade dopo $\sqrt{(n\pi)/2}$ passi (è una variante del famoso *paradosso dei compleanni*).

La prima idea è memorizzare l'intera sequenza e cercare duplicati. Questo metodo in media richiede un tempo $\mathcal{O}(\sqrt{n})$ e una memoria $\mathcal{O}(\sqrt{n})$. In realtà si può fare di meglio usando un algoritmo di tipo *lepre-tartaruga*: si parte dalla coppia (a_1, a_2) e si calcola (a_j, a_{2j}) finché $a_j = a_{2j}$. In altre parole, si scorre in parallelo la sequenza due volte, una (*lepre*) a velocità doppia dell'altra (*tartaruga*).

Perché questo metodo funziona? Indichiamo con λ la lunghezza della coda, e con μ quella del ciclo. Per ogni $i \geqslant \lambda$ e per ogni $k \geqslant 0$, si ha $a_i = a_{i+k\mu}$; in particolare, quando $i = k\mu \geqslant \lambda$, si ha $a_i = a_{2i}$.

Ora, sia p fattore primo (sconosciuto) di n. Utilizziamo l'algoritmo lepre-tartaruga per trovare i tale che $a_i \equiv a_{2i} \pmod{p}$. Poiché p è ignoto, la verifica viene compiuta calcolando il massimo comune divisore tra $a_i - a_{2i}$ e n: se il risultato è diverso da 1, abbiamo trovato un fattore primo di n. (Se dovesse capitare che $\mathrm{MCD}(a_i - a_{2i}, n) = n$, si ritenta con un'altra f.)

Algoritmo ρ DI POLLARD

Input: n intero composto, $f : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n$, a_0 valore iniziale

```
x := a_0
y := a_0
d := 1
while d = 1 do
    x := f(x)
    y := f(f(y))
    d := MCD(|x − y|, n)
end while
if d = n then
    return FALLIMENTO
else
    return d
end if
```

Output: d fattore primo non banale di n, oppure `FALLIMENTO`

Notiamo che questa implementazione dell'algoritmo richiede un tempo complessivo di computazione di $\mathcal{O}(\sqrt{p}) = \mathcal{O}(n^{1/4})$, e un uso limitato di memoria.

8.3 Algoritmo $p-1$ di Pollard

L'algoritmo $p-1$ di Pollard può essere utilizzato per trovare efficientemente fattori primi p di un intero composto n tali che $p-1$ sia B-liscio con B relativamente piccolo.

Definizione 8.8. Sia B un intero positivo. Un intero n è detto B-*liscio* se tutti i suoi fattori primi sono minori o uguali a B.

L'idea dietro l'algoritmo è la seguente. Consideriamo un intero B e sia $Q := \text{mcm}\{q^t \mid q \text{ primo}, q \leqslant B,\ q^t \leqslant n\}$. Se $q^\ell \leqslant n$, si ha $\ell \leqslant \left\lfloor \frac{\ln n}{\ln q} \right\rfloor$, quindi

$$Q = \prod_{q \leqslant B} q^{\left\lfloor \frac{\ln n}{\ln q} \right\rfloor}$$

dove il prodotto è su tutti i primi $q \leqslant B$. Se p è un fattore primo di n tale che $p-1$ sia B-liscio, allora $p-1 \mid Q$ per definizione di Q; quindi, per ogni a tale che $\text{MCD}(a,p) = 1$, per il piccolo teorema di Fermat $a^Q \equiv 1 \pmod{p}$. Detto $d := \text{MCD}(a^Q - 1, n)$, si ha che $p \mid d$, e se $d \neq 1$ e $d \neq n$, abbiamo trovato un fattore non banale di n.

Algoritmo $p-1$ **di** Pollard

Input: n intero composto, B fattore di liscezza

1: Scelta di $a \in \{2, \dots, n-1\}$
2: $d := \text{MCD}(a, n)$
3: **if** $d \geqslant 2$ **then**
4: **return** d
5: **end if**
6: **for all** q primi, $q \leqslant B$ **do**
7: $\ell := \left\lfloor \frac{\ln n}{\ln q} \right\rfloor$
8: $a := a^{q^\ell} \pmod{n}$
9: **end for**
10: $d := \text{MCD}(a-1, n)$
11: **if** $d = 1$ **or** $d = n$ **then**
12: **return** `FALLIMENTO`
13: **else**
14: **return** d
15: **end if**

Output: d fattore primo non banale di n, oppure `FALLIMENTO`

8.4 Metodo di Fermat

Questo metodo si basa sulla ben nota formula per la differenza di quadrati

$$x^2 - y^2 = (x + y)(x - y).$$

Quindi un numero che si scriva come $x^2 - y^2$ non è mai primo, a meno che uno dei fattori non sia 1.

Un numero dispari ammette sempre una scrittura come differenza di quadrati: se $n = ab$, allora

$$n = \left(\frac{a+b}{2}\right)^2 - \left(\frac{a-b}{2}\right)^2$$

dove le divisioni sono esatte, essendo a e b dispari.

Algoritmo METODO DI FERMAT

Input: n intero dispari

1: $a := \lceil \sqrt{n} \rceil$
2: $b := a^2 - n$
3: **while** b non è un quadrato **do**
4: $\quad a := a + 1$
5: $\quad b := a^2 - n$
6: **end while**
7: $p := a - \sqrt{b}$ ▷ oppure $p := a + \sqrt{b}$
8: **return** p

Output: p fattore primo di n, oppure $p = 1$

Nei casi peggiori, questo algoritmo potrebbe essere più lento del tentare tutte le possibili divisioni. In effetti, il metodo di Fermat diventa efficiente se i fattori di n sono vicini a $\sqrt{n}$.

8.5 Base di fattori e crivello quadratico

È possibile generalizzare il metodo di Fermat per ottenere un algoritmo di fattorizzazione più efficiente. Anziché cercare $n = x^2 - y^2$, risolviamo la congruenza

$$x^2 \equiv y^2 \pmod{n}$$

con $x \not\equiv \pm y \pmod{n}$. In tal caso, $\mathrm{MCD}(x + y, n)$ e $\mathrm{MCD}(x - y, n)$ forniscono fattori propri di n.

Da ora in avanti indicheremo con $[a]$ il rappresentante di $a \pmod n$ nell'intervallo $[-\frac{n}{2}, \frac{n}{2}]$.

Definizione 8.9. Una *base di fattori* è un insieme $\mathcal{B} = \{p_1, \dots, p_h\}$ di primi distinti, a parte p_1 che può essere -1. Diciamo che il quadrato di un intero b è un $\mathcal{B}$-*numero* (rispetto a un dato n) se $[b^2]$ può essere scritto come prodotto di numeri di $\mathcal{B}$.

Dati n e una base di fattori $\mathcal{B} = \{p_1, \dots, p_h\}$, associamo ad ogni $\mathcal{B}$-numero b^2 un vettore $\mathbf{b} \in (\mathbb{F}_2)^h$ nel seguente modo: scriviamo $[b^2] = \prod p_j^{\alpha_j}$ e fissiamo la j-esima componente $b_j = \alpha_j \pmod 2$.

Supponiamo di avere un insieme di $\mathcal{B}$-numeri b_i^2 tali che la somma dei corrispondenti vettori $\mathbf{b}_i$ sia nulla. Allora il prodotto di $[b_i^2]$ è un prodotto di potenze pari dei p_j. In altre parole, se $[b_i^2] = \prod p_j^{\alpha_{ij}}$, si ha

$$\prod [b_i^2] = \prod_{j=1}^{h} p_j^{\sum_i \alpha_{ij}}$$

dove tutti gli esponenti dei p_j a destra sono pari. Il membro di destra, quindi, è il quadrato di $\prod p_j^{\gamma_j}$, dove $\gamma_j = \frac{1}{2} \sum_i \alpha_{ij}$. Detti

$$b := [\prod_i b_i] \quad \text{e} \quad c := [\prod_j p_j^{\gamma_j}],$$

si ha che $b^2 \equiv c^2 \pmod n$.

Potrebbe capitare che $b \equiv \pm c \pmod n$, in tal caso scegliamo un diverso insieme di b_i.

Come scegliamo la base $\mathcal{B}$ e i b_i? Una prima idea è scegliere i primi h primi (o $h-1$, se usiamo $p_1 = -1$) e scegliere b_i a caso finché non ne troviamo abbastanza i cui quadrati siano $\mathcal{B}$-numeri. Un'altra strada consiste nel prendere i b_i tali che $[b_i^2]$ siano piccoli in valore assoluto (per esempio, b_i vicini a $\sqrt{kn}$ per multipli kn piccoli), e poi scegliere una base di fattori $\mathcal{B}$ formata da pochi primi piccoli, in modo che diversi b_i^2 siano $\mathcal{B}$-numeri.

Inoltre, dobbiamo scegliere abbastanza b_i in modo da essere sicuri di trovare tra i vettori $\mathbf{b}_i$ un sottoinsieme di vettori linearmente dipendenti su $\mathbb{F}_2$. Nel caso peggiore, ne servono $h+1$. Naturalmente, se non si vede ad occhio una relazione di dipendenza lineare, è possibile sistemare i vettori in una matrice e usare gli algoritmi di riduzione per righe.

Una variante del metodo della base di fattori è il *crivello quadratico*. Come base di fattori prendiamo

$$\mathcal{B} = \{2\} \cup \left\{ p \text{ primo}, 2 < p < P \;\middle|\; \left(\frac{n}{p}\right) = 1 \right\},$$

cioè i primi $p < P$ tali che n è un residuo quadratico modulo p, e 2 che va in $\mathcal{B}$ d'ufficio. P è una qualche costante scelta in qualche modo ottimale. L'insieme S in cui andiamo a cercare i $\mathcal{B}$-numeri è lo stesso usato per il metodo di Fermat:

$$S = \{t^2 - n \mid \lfloor \sqrt{n} \rfloor + 1 \leqslant t \leqslant \lfloor \sqrt{n} \rfloor + A\}$$

per qualche opportuna costante A. L'idea è setacciare l'insieme S con i primi $p \in \mathcal{B}$, costruendo una tabella della forma

S	primi di $\mathcal{B}$					
a_1	1	2	–	–	–	$\cdots$
a_2	–	1	–	–	2	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

sulle cui righe ci sono gli elementi $a_i \in S$ che sono $\mathcal{B}$-numeri, e nella riga che inizia per a_i, sotto a p_j, compare il massimo esponente di p_j che divide a_i. Una volta compilata la tabella, si può procedere come nel metodo della base di fattori, cercando righe che sommate diano un vettore di soli numeri pari.

Vediamo più in dettaglio.

1. Scegliamo le costanti P e A. Generalmente, vengono scelte di ordine $\mathcal{O}(e^{\sqrt{\ln n \ln\ln n}})$, con $P < A < P^2$.

2. Per $\lfloor \sqrt{n} \rfloor + 1 \leqslant t \leqslant \lfloor \sqrt{n} \rfloor + A$, scriviamo gli interi $t^2 - n$ in una colonna.

3. Costruiamo la base $\mathcal{B}$, prendendo 2 e, per ogni primo $2 < p < P$, calcolando $\left(\frac{n}{p}\right)$.

4. **Se p è dispari** (tratteremo a parte il caso $p = 2$), risolviamo l'equazione $t^2 \equiv n \pmod{p^\beta}$, per $\beta = 1, 2, \ldots$, ad esempio sollevando man mano le soluzioni "alla Hensel". Ci fermiamo al β tale che, per $\beta+1$, l'equazione non abbia soluzione nell'intervallo di interi $\lfloor \sqrt{n} \rfloor + 1 \leqslant t \leqslant \lfloor \sqrt{n} \rfloor + A$. Per tale β, siano t_1 e t_2 soluzioni dell'equazione $t^2 \equiv n \pmod{p^\beta}$ tali che $t_1 \equiv -t_2 \pmod{p^\beta}$ (t_1 e t_2 non sono necessariamente in $\lfloor \sqrt{n} \rfloor + 1 \leqslant t \leqslant \lfloor \sqrt{n} \rfloor + A$).

5. Facciamo scorrere β volte la colonna degli interi $t^2 - n$. Sotto la colonna p, al primo passaggio scriviamo 1 accanto ai valori $t^2 - n$ tali che $t \equiv t_1 \pmod{p}$; al secondo passaggio, cambiamo 1 con 2 accanto ai valori tali che $t \equiv t_1 \pmod{p^2}$; così via fino a p^β. Dopodiché ripetiamo con t_2 al posto di t_1. Il massimo intero che compare in questa colonna è β.

6. Mentre stiamo scorrendo la colonna, ogni volta che c'è un cambiamento (scriviamo 1, cambiamo 1 con 2,...), dividiamo per p il corrispondente $t^2 - n$, e teniamo traccia del risultato.

7. **Se** $p = 2$, se $n \not\equiv 1 \pmod 8$, semplicemente mettiamo 1 a $t^2 - n$ se t è dispari, e dividiamo il corrispondente $t^2 - n$ per 2. Se invece $n \equiv 1 \pmod 8$, risolviamo $t^2 \equiv n \pmod{2^\beta}$ e procediamo come nel caso di p dispari (l'unica differenza è che ci sono quattro diverse soluzioni t_1, t_2, t_3, t_4 modulo 2^β se $\beta \geqslant 3$).

8. Dopo aver finito i primi $p < P$, scartiamo dalla tabella tutti i $t^2 - n$ che non si sono ridotti a 1 dopo le ripetute divisioni: questi non erano $\mathcal{B}$-numeri.

9. A questo punto si procede come nel metodo della base di fattori.

8.6 Altri metodi

Accenneremo qui soltanto un altro paio di risultati sulla fattorizzazione di interi.

Un modo per trovare i b_i per il metodo della base di fattori consiste nell'usare le *frazioni continue*. Sia x numero reale; chiamiamo $a_0 = \lfloor x \rfloor$, e $x_0 = x - a_0$. Si ha $0 \leqslant x_0 < 1$; se $x_0 \neq 0$, $\frac{1}{x_0} > 1$, e definiamo $a_1 = \left\lfloor \frac{1}{x_0} \right\rfloor$, e $x_1 = \frac{1}{x_0} - a_1$. Possiamo ripetere il ragionamento, definendo $a_i = \left\lfloor \frac{1}{x_{i-1}} \right\rfloor$, e $x_i = \frac{1}{x_{i-1}} - a_i$. Non è difficile mostrare che il processo termina se e solo se x è razionale. Possiamo scrivere allora la *frazione continua* per x:

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \dots}}}.$$

Troncando la frazione continua a livello i, otteniamo un numero razionale, $\frac{b_i}{c_i}$, che chiameremo i-*esima convergenza* per la frazione continua. Si può dimostrare che, all'aumentare di i, le i-esime convergenze oscillano intorno a x, con oscillazioni sempre più piccole, fino a convergere a x per $i \longrightarrow \infty$.

Proposizione 8.10. *Sia n intero positivo non quadrato, e sia $\frac{b_i}{c_i}$ l'i-esima convergenza per la frazione continua di $\sqrt{n}$. Allora $[b_i^2] < 2\sqrt{n}$, dove $[b_i^2]$ indica il rappresentante di b_i^2 modulo n compreso tra $-\frac{n}{2}$ e $\frac{n}{2}$.*

Questa proposizione (che non dimostreremo) ci fornisce un modo per trovare una sequenza di b_i utili per il metodo della base di fattori. È sufficiente, infatti,

prendere i numeratori delle convergenze della frazione continua per $\sqrt{n}$. Tra l'altro, non dobbiamo neppure calcolare l'intera convergenza, ma solo il numeratore, e per di più modulo n.

Ad oggi, il più efficiente algoritmo di fattorizzazione è il *crivello dei campi di numeri generale* (GNFS, dall'inglese *General Number Field Sieve*). Accenniamo brevemente le motivazioni che hanno portato allo sviluppo di questo algoritmo.

Nel crivello quadratico, cercavamo dei numeri b_i tali che $b_i^2 - n$ fossero dei $\mathcal{B}$-numeri. In altre parole, lavoravamo con il polinomio $f(T) = T^2 - n$. Questo polinomio può essere visto come omomorfismo di anelli $f : \mathbb{Z} \longrightarrow \mathbb{Z}_n$, e mappa un quadrato di $\mathbb{Z}$ in un quadrato di $\mathbb{Z}_n$. Più in generale, supponiamo di avere un anello R e un omomorfismo di anelli $\varphi : R \longrightarrow \mathbb{Z}_n$. Se $\beta \in R$ è tale che $\varphi(\beta^2) = y^2 \pmod n$, e $\varphi(\beta) = x \pmod n$, allora

$$x^2 \equiv \varphi(\beta)^2 \equiv \varphi(\beta^2) \equiv y^2 \pmod n.$$

Quindi, se troviamo un elemento di R che è un quadrato perfetto, e viene mappato da φ in un quadrato perfetto, allora applicando φ si ottiene una differenza di quadrati. Come anello R viene preso l'anello $\mathbb{Z}[\theta]$ delle combinazioni lineari formali a coefficienti in $\mathbb{Z}$ degli elementi $\{1, \theta, \ldots, \theta^{d-1}\}$, dove θ è radice di un polinomio monico irriducibile di grado d in $\mathbb{Z}[T]$.

Per produrre differenze di quadrati ci basiamo sulla seguente proposizione.

Proposizione 8.11. *Dato $f(T) \in \mathbb{Z}[T]$ polinomio irriducibile e monico, una sua radice $\theta \in \mathbb{C}$, e un intero $m \in \mathbb{Z}_n$ tale che $f(m) \equiv 0 \pmod n$, la mappa*

$$\varphi : \mathbb{Z}[\theta] \longrightarrow \mathbb{Z}_n$$

tale che $\varphi(1) = 1$ e $\varphi(\theta) = m$ è un omomorfismo di anelli suriettivo.

Supponiamo di trovare un insieme $U \subseteq \mathbb{Z} \times \mathbb{Z}$ tale che

$$\prod_{(a,b)\in U} (a + b\theta) = \beta^2 \quad \text{e} \quad \prod_{(a,b)\in U} (a + bm) = y^2$$

con $\beta \in \mathbb{Z}[\theta]$ e $y \in \mathbb{Z}$. Allora, applicando φ dato dalla proposizione sopra, e detto $\varphi(\beta) = x \in \mathbb{Z}_n$, si ha

$$\begin{aligned} x^2 \equiv \varphi(\beta)^2 \equiv \varphi(\beta^2) &\equiv \varphi\left(\prod_{(a,b)\in U} (a + b\theta)\right) \\ &\equiv \prod_{(a,b)\in U} \varphi(a + b\theta) \equiv \prod_{(a,b)\in U} (a + bm) \equiv y^2 \pmod n. \end{aligned}$$

L'algoritmo GNFS si occupa proprio di trovare questo U, utilizzando contemporaneamente basi di fattori "algebriche" per $\mathbb{Z}[\theta]$ e "razionali" per $\mathbb{Z}$.

Capitolo 9

Il logaritmo discreto

L'altro problema computazionalmente difficile che tratteremo è il logaritmo discreto: dati G gruppo ciclico, g suo generatore, e $a \in G$, trovare $b \in \mathbb{Z}$ tale che $g^b = a$. Tale b prende il nome di *logaritmo discreto* in base g di a, e scriveremo $b = \log_g a$. Naturalmente è sempre possibile partire da g e calcolare $g^2, g^3, \ldots$ finché non si arriva ad a, ma questo metodo richiede $\mathcal{O}(n)$ tempo di calcolo (n è l'ordine del gruppo), che è proibitivo.

Nei protocolli crittografici generalmente si sceglie $G = (\mathbb{Z}_p)^*$.

9.1 L'algoritmo *baby-step giant-step*

L'idea dietro l'algoritmo *baby-step giant-step* è molto semplice: è possibile suddividere un gruppo di ordine n in macroblocchi di ordine circa $\sqrt{n}$, ciascuno contenente circa $\sqrt{n}$ elementi.

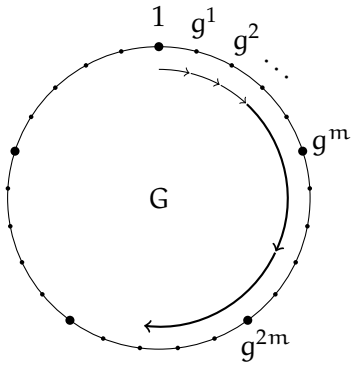


Figura 9.1: Rappresentazione grafica dell'algoritmo *baby-step giant-step*.

In altre parole, detto $m = \lceil \sqrt{n} \rceil$, se $a = g^b$, possiamo scrivere $b = mi + j$ con $0 \leqslant i, j < m$. Dunque $ag^{-im} = g^j$, per qualche $i, j = 0, \ldots, m-1$.

Questo porta al seguente algoritmo:

1. *Baby-step*: con le stesse notazioni di prima, costruiamo una tabella con (j, g^j) come elementi al variare di $j = 0, \ldots, m-1$.

2. *Giant-step*: calcoliamo ag^{-im} per $i = 0, \ldots, m-1$, cercando ad ogni passaggio se ag^{-im} sta nella tabella del *baby-step*. Per quanto visto sopra, troviamo un j tale che $ag^{-im} = g^j$. L'algoritmo restituisce, quindi, $mi + j$.

L'algoritmo è reso più efficiente dal poter ordinare la tabella del *baby-step*, per agevolare la ricerca all'interno di essa. In effetti, questo algoritmo richiede un tempo massimo di $\mathcal{O}(\sqrt{n}\ln(\sqrt{n}))$, e $\mathcal{O}(\sqrt{n})$ di spazio in memoria.

9.2 Algoritmo ρ di Pollard per il logaritmo discreto

L'algoritmo ρ di Pollard descritto nella sezione 8.2 può essere adattato per risolvere il problema del logaritmo discreto. Per risolvere $g^b = a$, l'algoritmo cerca interi γ, α, $\tilde{\gamma}$, $\tilde{\alpha}$ tali che $g^\gamma a^\alpha = g^{\tilde{\gamma}} a^{\tilde{\alpha}}$, e quindi trova b risolvendo la congruenza $(\tilde{\alpha} - \alpha)b \equiv \gamma - \tilde{\gamma} \pmod{n}$. In particolare, l'algoritmo usa un approccio di tipo lepre-tartaruga per trovare una collisione in una sequenza $x_i = g^{\gamma_i} a^{\alpha_i}$.

Partizioniamo G in tre sottoinsiemi G_0, G_1, G_2 circa delle stesse dimensioni, tali che $1 \notin G_1$. Definiamo

$$\begin{aligned} f : G &\longrightarrow G \\ x &\longmapsto \begin{cases} ax & \text{se } x \in G_0 \\ x^2 & \text{se } x \in G_1 \\ gx & \text{se } x \in G_2 \end{cases} \end{aligned}$$

e la successione delle iterate

$$\begin{cases} x_0 = 1 \\ x_{i+1} = f(x_i). \end{cases}$$

Questa successione induce le due successioni sugli esponenti

$$\begin{cases} \gamma_0 = 0 \\ \gamma_{i+1} = \begin{cases} \gamma_i & \text{se } x_i \in G_0 \\ 2\gamma_i & \text{se } x_i \in G_1 \\ \gamma_i + 1 & \text{se } x_i \in G_2 \end{cases} \end{cases} \quad \text{e} \quad \begin{cases} \alpha_0 = 0 \\ \alpha_{i+1} = \begin{cases} \alpha_i + 1 & \text{se } x_i \in G_0 \\ 2\alpha_i & \text{se } x_i \in G_1 \\ \alpha_i & \text{se } x_i \in G_2. \end{cases} \end{cases}$$

Nel listato dell'algoritmo, per comodità di scrittura, scriviamo

$$\varphi : G \times \mathbb{Z} \longrightarrow \mathbb{Z} \qquad (x, m) \longmapsto \begin{cases} m & \text{se } x \in G_0 \\ 2m & \text{se } x \in G_1 \\ m+1 & \text{se } x \in G_2 \end{cases}$$

$$\psi : G \times \mathbb{Z} \longrightarrow \mathbb{Z} \qquad (x, m) \longmapsto \begin{cases} m+1 & \text{se } x \in G_0 \\ 2m & \text{se } x \in G_1 \\ m & \text{se } x \in G_2 \end{cases}$$

dove tutte le operazioni sono fatte modulo n (l'ordine di G).

Algoritmo ρ **di Pollard per il logaritmo discreto**

Input: g generatore di G, $a \in G$

$\alpha_0 := 0$
$\gamma_0 := 0$
$x_0 := 1 \in G$
$i := 1$
while $x_i \neq x_{2i}$ **do**
    $x_i := f(x_{i-1})$
    $\gamma_i := \varphi(x_{i-1}, \gamma_{i-1})$
    $\alpha_i := \psi(x_{i-1}, \alpha_{i-1})$
    $x_{2i} := f(f(x_{2i-2}))$
    $\gamma_{2i} := \varphi(f(x_{2i-2}), \varphi(x_{2i-2}, \gamma_{2i-2}))$
    $\alpha_{2i} := \psi(f(x_{2i-2}), \psi(x_{2i-2}, \alpha_{2i-2}))$
    $i := i + 1$
end while
if $\alpha_i = \alpha_{2i}$ **then**
    return `FALLIMENTO`
else
    $b := (\alpha_i - \alpha_{2i})^{-1}(\gamma_{2i} - \gamma_i) \pmod{n}$
    return b
end if

Output: b tale che $g^b = a$, oppure `FALLIMENTO`

9.3 Algoritmo di Pohlig-Hellman

L'algoritmo che presentiamo ora è particolarmente efficiente se l'ordine del gruppo si fattorizza con primi piccoli. Supponiamo che $n = \prod_{i=1}^{r} p_i^{e_i}$, e $b = \log_g a$. Innanzitutto, vogliamo determinare $b_i \equiv b \pmod{p_i^{e_i}}$ per poi rimontare la

soluzione con il teorema cinese del resto. Ogni intero b_i è ottenuto calcolando le cifre ℓ_j, $j = 0, \dots, e_i - 1$ della sua rappresentazione p_i-aria

$$b_i = \sum_{j=0}^{e_i-1} \ell_j p_i^j, \quad 0 \leqslant \ell_j \leqslant p_i - 1.$$

Algoritmo POHLIG-HELLMAN

Input: g generatore di G, $a \in G$

1: Fattorizzazione di $n = \prod p_i^{e_i}$
2: **for** $i := 1$ **to** r **do**
3: $\quad q := p_i$ ▷ per semplificare la notazione
4: $\quad e := e_i$ ▷ come sopra
5: $\quad \gamma := 1$
6: $\quad \ell_{-1} := 0$
7: $\quad \tilde{g} := g^{\frac{n}{q}}$
8: $\quad$ **for** $j := 0$ **to** $e - 1$ **do**
9: $\quad\quad \gamma := \gamma g^{\ell_{j-1} p^{j-1}}$
10: $\quad\quad \tilde{a} := (a\gamma^{-1})^{\frac{n}{q^{j+1}}}$
11: $\quad\quad \ell_j := \log_{\tilde{g}} \tilde{a}$ ▷ usando, per esempio, *baby-step giant-step*
12: $\quad$ **end for**
13: $\quad b_i := \ell_0 + \ell_1 q + \cdots + \ell_{e-1} q^{e-1}$
14: **end for**
15: Calcolo di b a partire dai b_i tramite teorema cinese del resto
16: **return** b

Output: b tale che $g^b = a$

Perché l'algoritmo funziona? Alla j-esima iterazione (posto $q := p_i$ ed $e := e_i$) si ha $\gamma = g^{\ell_0 + \ell_1 q + \cdots + \ell_{j-1} q^{j-1}}$, e dunque

$$\begin{aligned} \tilde{a} = (a\gamma^{-1})^{n/q^{j+1}} &= (g^{b - \ell_0 - \ell_1 q - \cdots - \ell_{j-1} q^{j-1}})^{n/q^{j+1}} = \\ &\overset{(1)}{=} (g^{n/q^{j+1}})^{kq^e + b_i - \ell_0 - \ell_1 q - \cdots - \ell_{j-1} q^{j-1}} = \\ &\overset{(2)}{=} (g^{n/q^{j+1}})^{\ell_j q^j + \cdots + \ell_{e-1} q^{e-1}} = (g^{n/q})^{\ell_j + \cdots + \ell_{e-1} q^{e-1-j}} \overset{(3)}{=} \tilde{g}^{\ell_j} \end{aligned}$$

dove

(1) abbiamo esplicitato $b = b_i + kq^e$ per un certo k;

(2) ovviamente, $(g^{n/q^{j+1}})^{kq^e} = 1$ per ogni $j = 0, \dots, e-1$ e per ogni $k \in \mathbb{Z}$;

(3) gli altri termini sono potenze di $g^n = 1$.

9.4 Ancora base di fattori

Una tabella come quella costruita per il crivello quadratico (vedi sezione 8.5) può essere utile anche per risolvere il problema del logaritmo discreto.

Sia G gruppo ciclico con generatore g. Scegliamo un piccolo numero di elementi "irriducibili" (sostanzialmente, che non si scrivono come prodotto di elementi più "piccoli"; per $(\mathbb{Z}_p)^*$, si può scegliere tra i primi q che dividono $p-1$). Chiamiamo *base di fattori* l'insieme di questi elementi.

L'idea principale è quella di risolvere il logaritmo discreto per gli elementi della base di fattori, e utilizzare poi tali logaritmi per determinare quello di un elemento generico. Per fare ciò, cerchiamo potenze del generatore che si fattorizzino completamente in elementi della base di fattori $\mathcal{B}$ (in un certo senso, che siano dei $\mathcal{B}$-numeri). Tramite queste, ricaviamo un sistema lineare in cui le incognite sono proprio i logaritmi della base di fattori.

In particolare, se $\mathcal{B} = \{p_1, \ldots, p_h\}$, e otteniamo

$$g^{r_i} = \prod_{j=1}^{h} p_j^{t_{ij}},$$

ricaviamo la relazione lineare (a coefficienti in $\mathbb{Z}$)

$$r_i = \sum_{j=1}^{h} t_{ij} x_j$$

dove le incognite sono $x_j = \log_g p_j$. Quando abbiamo abbastanza relazioni, risolviamo il sistema lineare, ricordando che stiamo lavorando modulo n, quindi potrebbe essere necessario spezzare il sistema e rimontarlo tramite teorema cinese del resto, per evitare di incappare in zero-divisori.

Una volta noti $x_j = \log_g p_j$, prendiamo a caso una potenza s di g e vediamo se ag^s si può scrivere come prodotto di termini della base di fattori. Se otteniamo

$$ag^s = \prod_{j=1}^{h} p_j^{t_j},$$

allora possiamo ricavare $\log_g a = (\sum t_j x_j) - s$.

Capitolo 10

Principali crittosistemi a chiave pubblica

Presentiamo in questo capitolo tre crittosistemi basati sui problemi presentati in precedenza. In particolare, i protocolli Diffie-Hellman ed ElGamal sfruttano il logaritmo discreto, mentre RSA usa la fattorizzazione.

10.1 Il protocollo Diffie-Hellman

Questo protocollo è stato presentato da Whitfield Diffie e Martin Hellman in un articolo del 1976, che segna l'inizio della crittografia a chiave pubblica. Più che un crittosistema a chiave pubblica, è un protocollo di *scambio delle chiavi*: i due interlocutori si scambiano attraverso un canale pubblico delle informazioni che, unitamente ad altre informazioni tenute private, consentono loro di costruire una chiave *simmetrica* comune, che può essere usata quindi per la codifica e decodifica di un messaggio.

Chiamiamo Alice e Bob i due interlocutori.

1. Alice e Bob scelgono di comune accordo un primo p, e un generatore g di $(\mathbb{Z}_p)^*$.

2. Alice sceglie un numero segreto a, Bob un numero segreto b.

3. Alice invia a Bob $A := g^a \pmod p$, Bob invia $B := g^b \pmod p$.

4. Alice calcola $B^a = (g^b)^a \pmod p$ e Bob $A^b = (g^a)^b \pmod p$. Questi numeri sono uguali e sono la chiave segreta condivisa da Alice e Bob.

Quindi sono accessibili a tutti p, g, g^a e g^b. Un eventuale malintenzionato dovrebbe ricavare g^{ab} a partire da g^a e g^b. Se avesse un algoritmo efficiente per il logaritmo discreto, potrebbe ottenere a e b da g^a e g^b, e poi calcolare g^{ab}. Finché il problema del logaritmo discreto è intrattabile, Diffie-Hellman resta sicuro.

In effetti, non siamo sicuri che non si possa calcolare g^{ab} a partire da g^a e g^b, senza usare il logaritmo discreto. Al giorno d'oggi non si conoscono algoritmi in quella direzione: il logaritmo discreto è il meglio che possiamo fare.

10.2 Il protocollo ElGamal

A partire dall'idea di Diffie ed Hellman, l'egiziano Taher Elgamal ha descritto un protocollo di cifratura a chiave asimmetrica, anch'esso basato dunque sul logaritmo discreto.

1. Alice sceglie un primo p e un generatore $g \in (\mathbb{Z}_p)^*$, dopodiché sceglie a e calcola $A := g^a \pmod p$. La chiave pubblica è (p, g, A), mentre quella privata è a.

2. Quando Bob vuole mandare un messaggio m ad Alice, sceglie un intero b e calcola $B := g^b \pmod p$. Quindi calcola la chiave di cifratura $K = A^b = g^{ab}$, cifra il messaggio calcolando $m' = Km$ e invia ad Alice (B, m').

3. Per decifrare, Alice calcola la chiave K come B^a, quindi recupera il messaggio calcolando $K^{-1}m' = m$.

Notiamo che la chiave di cifratura viene generata nel momento in cui Bob vuole mandare un messaggio, a partire dalle informazioni pubbliche di Alice e da un parametro scelto segretamente da Bob.

10.3 Il protocollo RSA

Nel 1978 tre ricercatori del MIT, Ron Rivest, Adi Shamir e Leonard Adleman, descrissero un algoritmo di cifratura a chiave pubblica basato sulla difficoltà di fattorizzare in $\mathbb{Z}$. L'algoritmo prende il nome RSA dalle iniziali dei tre ricercatori. Nella descrizione dell'algoritmo, denoteremo con $\varphi(n)$ la funzione φ di Eulero.

1. Innanzitutto, Alice genera la sua coppia di chiavi. Per fare ciò, sceglie opportunamente due numeri primi p e q e ne fa il prodotto $n = pq$. Quindi calcola $\varphi(n) = (p-1)(q-1)$, e sceglie un numero $1 < e < \varphi(n)$ tale che

$\mathrm{MCD}(e, \varphi(n)) = 1$. La coppia (n, e) è la *chiave pubblica* di Alice. Infine Alice calcola d tale che $de \equiv 1 \pmod{\varphi(n)}$, che servirà per decifrare. La *chiave privata* è formata da (p, q, d).

2. Se Bob vuole mandare un messaggio ad Alice, innanzitutto lo converte in un intero $0 < m < n$ (con $\mathrm{MCD}(m, n) = 1$), quindi calcola $c = m^e \pmod{n}$ e invia c ad Alice.

3. Per recuperare il messaggio, Alice deve solo calcolare $c^d \equiv m^{ed} \equiv m \pmod{n}$, dove quest'ultima uguaglianza segue dal teorema di Eulero.[1]

La sicurezza del protocollo RSA discende dal fatto che per ricostruire d è necessario conoscere $\varphi(n)$, ma questo è tanto difficile quanto fattorizzare n.

Proposizione 10.1. *Sia $n = pq$ prodotto di due primi. Conoscere p e q è equivalente a conoscere $\varphi(n)$.*

Dimostrazione. $\boxed{\Rightarrow}$ Sapere p e q permette di calcolare $\varphi(n) = (p-1)(q-1)$.
$\boxed{\Leftarrow}$ Da $n = pq$ e $\varphi(n) = (p-1)(q-1)$ si ricava che le radici del polinomio

$$X^2 - (n + 1 - \varphi(n))X + n$$

sono proprio p e q. □

[1] Il *teorema di Eulero* generalizza il piccolo teorema di Fermat, affermando che $a^{\varphi(n)} \equiv 1 \pmod{n}$ se $\mathrm{MCD}(a, n) = 1$.

Capitolo 11

Curve ellittiche

Recentemente hanno trovato importanza in crittografia alcuni oggetti legati principalmente alla geometria algebrica e alla teoria dei numeri, le cosiddette *curve ellittiche*. Il motivo principale è che le curve ellittiche forniscono una notevole quantità di gruppi finiti su cui è particolarmente facile lavorare, perché molto ricchi di struttura.

Definizione 11.1. Sia $\mathbb{K}$ un campo. Un'equazione della forma

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \tag{11.1}$$

dove $a_1, \ldots, a_6 \in \mathbb{K}$ è detta *equazione di Weierstrass*.

Nel seguito indicheremo con $F(X, Y, Z) \in \mathbb{K}[X, Y, Z]$ il polinomio che definisce un'equazione di Weierstrass. In altre parole, scriviamo (11.1) come

$$F(X, Y, Z) = 0.$$

Definizione 11.2. Dato un campo $\mathbb{K}$, una *curva ellittica* definita su $\mathbb{K}$, indicata con $E/_{\mathbb{K}}$, è l'insieme delle soluzioni in $\mathbb{P}^2(\mathbb{K})$ di un'equazione di Weierstrass, cioè

$$E/_{\mathbb{K}} := \{[x : y : z] \in \mathbb{P}^2(\mathbb{K}) \mid F(x, y, z) = 0\}.$$

Essendo F polinomio omogeneo di terzo grado, una curva ellittica è ben definita. Nel seguito, quando non ci sono ambiguità, indicheremo una curva ellittica anche solo con E, senza esplicitare il campo $\mathbb{K}$ su cui è definita.

Mettiamoci adesso nella carta $U_z := \{z \neq 0\}$, che identificheremo con un piano affine $\mathbb{A}^2(\mathbb{K})$, e studiamo separatamente $E \cap U_z$ e $E \cap \{z = 0\}$.

Nella carta U_z, possiamo deomogeneizzare rispetto alla coordinata z, applicando

$$(x, y, z) \longmapsto \left(\frac{x}{z}, \frac{y}{z}, 1\right)$$

che ci porta all'*equazione di Weierstrass non omogenea*

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{11.2}$$

dove abbiamo ribattezzato $x := x/z$ e $y := y/z$. Indicheremo con $f(X, Y) \in \mathbb{K}[X, Y]$ il polinomio che definisce l'equazione (11.2).

Quindi, in U_z possiamo mettere in corrispondenza biunivoca le soluzioni in $\mathbb{P}^2(\mathbb{K})$ di (11.1) con le soluzioni in $\mathbb{A}^2(\mathbb{K})$ di (11.2).

Cosa succede se $z = 0$? Sostituendo in (11.1), otteniamo $x^3 = 0$ e nessuna restrizione sulla y (se non la ovvia $y \neq 0$ affinché il punto del piano proiettivo sia definito). In altre parole,

$$E/_{\mathbb{K}} \cap \{z = 0\} = \{[0 : 1 : 0]\}.$$

Il punto $O := [0 : 1 : 0]$ è detto *punto all'infinito* della curva ellittica.

Osservazione. Se $\mathrm{char}(\mathbb{K}) \neq 2$, l'equazione (11.2) si può semplificare completando il quadrato; in effetti, la sostituzione

$$y \longmapsto \frac{y - a_1x - a_3}{2}$$

porta l'equazione nella forma

$$y^2 = 4x^3 + b_2x^2 + 2b_4x + b_6, \tag{11.3}$$

dove $b_2 := a_1^2 + 4a_4$, $b_4 := 2a_4 + a_1a_3$ e $b_6 := a_3^2 + 4a_6$. Se inoltre $\mathrm{char}(\mathbb{K}) \neq 2, 3$ la (11.3) si può ulteriormente semplificare in

$$y^2 = x^3 - 27c_4x - 54c_6,$$

con $c_4 := b_2^2 - 24b_4$ e $c_6 := -b_2^3 + 36b_2b_4 - 216b_6$, tramite il cambiamento di coordinate

$$(x, y) \longmapsto \left(\frac{x - 3b_2}{36}, \frac{y}{108}\right).$$

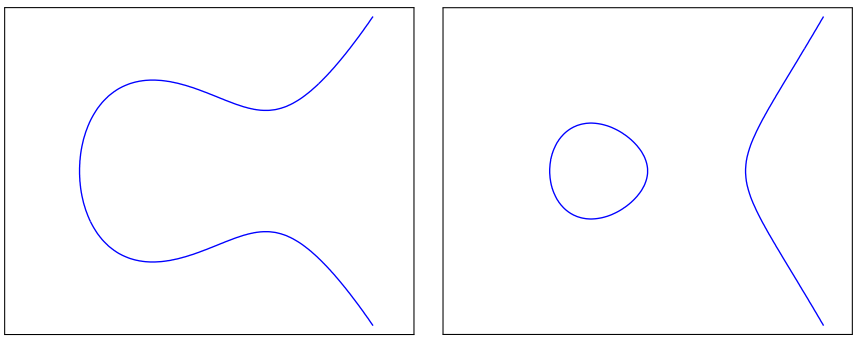


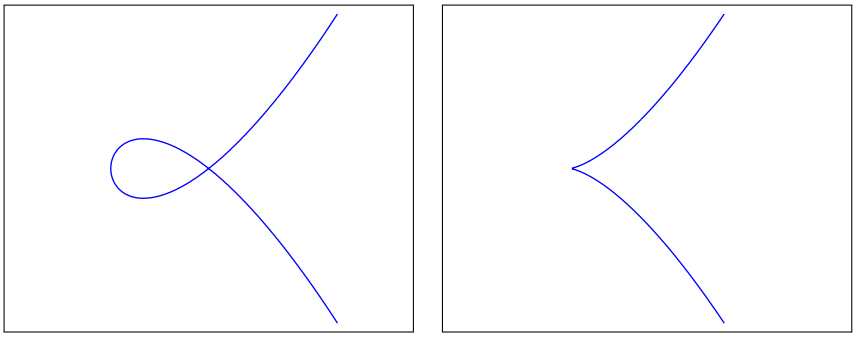
Figura 11.1: I due possibili grafici (a meno di trasformazioni affini) di una curva ellittica definita su $\mathbb{R}$.

Definizione 11.3. Sia $E/_{\mathbb{K}}$ una curva ellittica definita da un'equazione di Weierstrass $F(X, Y, Z) = 0$. Un punto $P \in \mathbb{P}^2(\mathbb{K})$ è detto *punto singolare* se

$$\frac{\partial F}{\partial X}(P) = \frac{\partial F}{\partial Y}(P) = \frac{\partial F}{\partial Z}(P) = 0.$$

La curva si dice *singolare* se ha almeno un punto singolare, *non singolare* altrimenti.

In un punto singolare, la curva non ha una tangente ben definita, e questo impedisce la buona definizione della legge di gruppo. Nel resto del capitolo, se non specificato, la curva ellittica si intende non singolare.



(a) Nodo. (b) Cuspide.

Figura 11.2: I due tipi di punto singolare.

Un'ultima, importante osservazione. Se la curva ellittica è definita su un campo finito, logicamente avrà un numero finito di punti. Una stima di quanti siano effettivamente questi punti è data dal seguente teorema, di cui omettiamo la dimostrazione.

Teorema 11.4 (Hasse). *Sia $E/_{\mathbb{F}_q}$ una curva ellittica definita sul campo finito $\mathbb{F}_q$. Allora*

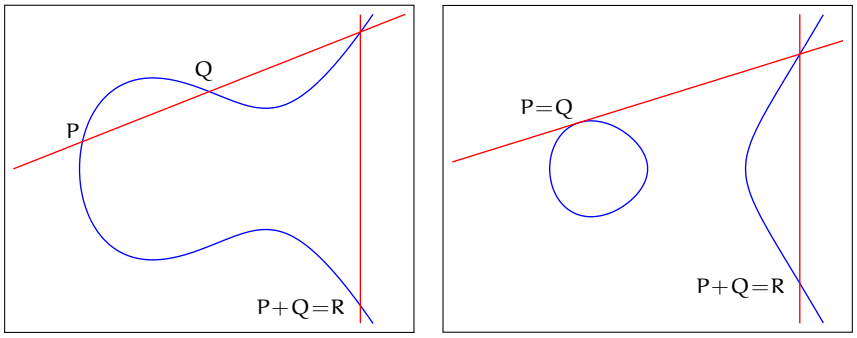
$$|\#E - (q + 1)| \leqslant 2\sqrt{q}.$$

11.1 Legge di gruppo

Sia $E/_{\mathbb{K}}$ una curva ellittica, e A un suo punto fissato. Siano P e Q due suoi punti; sia ℓ la retta che passa per P e Q, e sia R' il terzo punto di intersezione di ℓ con E. Sia ora ℓ' la retta passante per A e R', e sia R il terzo punto di intersezione di ℓ' con E. Definiamo

$$P + Q := R.$$

Si può dimostrare che la scelta del punto A è del tutto arbitraria. Normalmente viene scelto $A = O$: in questo modo i conti risultano notevolmente ridotti.



(a) Caso $P \neq Q$. (b) Caso $P = Q$.

Figura 11.3: Somma di punti.

Teorema 11.5. $(E, +)$ *è un gruppo abeliano, con* O *come elemento neutro.*

Dimostrazione. Mostriamo che $(E, +)$ soddisfa gli assiomi di gruppo.

Chiusura. Scrivendo esplicitamente le formule per il calcolo delle coordinate della somma di punti, si può notare che esse coinvolgono solo operazioni di campo. Quindi il punto risultante sta ancora in $E/_{\mathbb{K}}$.

Elemento neutro. Il punto all'infinito O è l'elemento neutro della somma. Dato un punto P, la retta passante per P e per O incontra la curva nell'unico altro punto P' che ha la stessa ascissa di P. La retta passante per P' e per O, chiaramente, incontra la curva in P. Quindi $P + O = P$.

Elemento inverso. Procedendo come per l'elemento neutro, si vede che l'inverso di un punto P è il punto P' con la stessa ascissa di P.

Commutatività. La retta che passa per P e per Q è la stessa che passa per Q e P.

Associatività. La dimostrazione dell'associatività è lunga e richiede strumenti avanzati di algebra, e pertanto verrà omessa.

□

Nel caso in cui $P = Q$, si prende la retta tangente alla curva in P. Questa definizione di somma fa sì che per P, Q e R allineati si abbia

$$P + Q + R = O.$$

Un paio di semplici conti mostrano che, in effetti, le formule esplicite per calcolare $R = (x_3, y_3)$, somma di $P = (x_1, y_1)$ e $Q = (x_2, y_2)$, sono

$$\begin{cases} x_3 = m^2 + a_1 m - a_2 - x_1 - x_2 \\ y_3 = -(m + a_1)x_3 - q - a_3, \end{cases}$$

dove $y = mx + q$ è la retta passante per P e Q (tangente se $P = Q$), con

$$m = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & \text{se } P \neq Q \\ \dfrac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3} & \text{se } P = Q \end{cases}$$

$$q = y_1 - mx_1.$$

11.2 Qualche applicazione crittografica

Vogliamo innanzitutto calcolare il numero di punti di una curva ellittica: il teorema di Hasse ci fornisce solo una stima. Un'idea per calcolare quanti punti ha la curva può essere calcolare l'ordine dei suoi punti, ad esempio con un algoritmo "alla ρ di Pollard": per trovare l'ordine di P, consideriamo la successione

$$\begin{cases} P_0 = P \\ P_{i+1} = P_i + P \end{cases}$$

e cerchiamo il primo indice per cui si abbia $P_{2i} = P_i$. La nostra speranza è che nell'intervallo fornito dal teorema di Hasse cada un solo multiplo dell'ordine trovato. Eventualmente si possono calcolare più ordini e verificare che cadano nella stima di Hasse solo i multipli del loro mcm.

Il problema del logaritmo discreto è definibile su una curva ellittica come: dati due punti $P, Q \in E/_{\mathbb{K}}$, determinare il più piccolo k tale che $Q = kP$, dove kP significa $P + \cdots + P$ k volte. Di conseguenza, è possibile adattare lo scambio di chiavi alla Diffie-Hellman sulle curve ellittiche.

1. Alice e Bob scelgono un campo finito $\mathbb{F}_q$ e una curva ellittica $E/_{\mathbb{F}_q}$; inoltre scelgono un punto $P \in E$.

2. Alice sceglie $a \in \mathbb{Z}$ e trasmette aP a Bob. Dal canto suo, Bob sceglie $b \in \mathbb{Z}$ e trasmette bP ad Alice.

3. I due condividono l'informazione $abP = a(bP) = b(aP)$.

Un metodo per scegliere una curva ellittica è fissare prima $P = (x, y) \in (\mathbb{F}_q)^2$, scegliere $a \in \mathbb{F}_q$ e porre $b := y^2 - x^3 - ax$. In questo modo si è trovata una curva ellittica $y^2 = x^3 + ax + b$ passante per P.

Sulle curve ellittiche è possibile sviluppare un test di primalità simile al test di Pocklington (vedi sezione 7.4), che va sotto il nome di *algoritmo di Goldwasser-Kilian*. Quelle che costruiremo in effetti saranno delle *pseudo-curve ellittiche*, nel senso che tratteremo oggetti definiti da equazioni di Weierstrass su $\mathbb{Z}_n$, con n non necessariamente primo.

Proposizione 11.6. *Siano n un intero positivo, $a, b \in \mathbb{Z}_n$, e sia*

$$E := \{(x, y) \in (\mathbb{Z}_n)^2 \mid y^2 \equiv x^3 + ax + b \pmod{n}\} \cup \{O\}$$

dove O è un simbolo che denota il "punto all'infinito". Sia inoltre m un intero. Supponiamo che esista un primo q che divide m e tale che $q > (n^{1/4} + 1)^2$. Se esiste $P \in E$ tale che $mP = O$ e $\frac{m}{q}P \neq O$, allora n è primo.

Dimostrazione. Supponiamo che n non sia primo, e sia $p \leqslant \sqrt{n}$ un primo che divide n. Sia $E'/_{\mathbb{Z}_p}$ la curva ellittica definita su $\mathbb{Z}_p$ dalla stessa equazione di E, e sia m' l'ordine del gruppo dei punti di E'. Dal teorema di Hasse

$$m' \leqslant p + 1 + 2\sqrt{p} = (\sqrt{p} + 1)^2 \leqslant (\sqrt[4]{n} + 1)^2 < q,$$

quindi q e m' sono coprimi. Allora esiste u tale che $uq \equiv 1 \pmod{m'}$.

Sia ora $P' \in E'$ il punto P letto modulo p. Si ha ancora $mP' = O$ e $\frac{m}{q}P' \neq O$, basta ripetere il conto lavorando modulo p anziché modulo n. Ma ora

$$\frac{m}{q}P' = uq\frac{m}{q}P' = umP' = O,$$

in contraddizione con $\frac{m}{q}P' \neq O$. □

Notiamo che le formule per la somma di punti di una curva ellittica richiedono alcune divisioni. Se la curva ellittica è definita in un campo, non ci sono problemi; ma considerando una pseudo-curva ellittica su $\mathbb{Z}_n$, il calcolo di kP a un certo punto potrebbe fallire. Questo è un vantaggio per noi: il calcolo fallisce quando si è trovato uno zero-divisore in $\mathbb{Z}_n$, sostanzialmente un divisore di n. Questa è l'idea alla base dell'algoritmo di fattorizzazione su curve ellittiche di Lenstra, simile all'algoritmo $p - 1$ di Pollard (vedi sezione 8.3). Il vantaggio di questo algoritmo è che non è *one-shot*: in caso di fallimento, possiamo sempre cambiare punto e/o curva.

Più in dettaglio, il calcolo di kP richiede la divisione tra classi di resto modulo n, che può essere compiuta tramite l'algoritmo euclideo esteso. In particolare,

la divisione per un certo v modulo n richiede il calcolo di $\text{MCD}(v, n)$. Ora, se $\text{MCD}(v, n) = 1$, non ci sono problemi, perché v è invertibile. Anche se $v \equiv 0 \pmod{n}$ non ci sono problemi, perché il risultato della somma sarà il punto all'infinito. Se invece $\text{MCD}(v, n) \neq 1$ oppure n, la divisione non può essere svolta, e abbiamo trovato un fattore non banale di n.

1. Scegliamo un'equazione del tipo $y^2 = x^3 + ax + b$ in $\mathbb{Z}_n$, e un punto P.
2. Calcoliamo $eP \in E/_{\mathbb{Z}_n}$, dove e è prodotto di molti numeri piccoli, per esempio prodotto di potenze di primi piccoli, oppure $B!$ per qualche B non troppo grande: si può fare efficientemente calcolando $(2!)P$, $(3!)P = 3(2!)P$, e così via.
3. - Se siamo riusciti a compiere tutte le operazioni, proviamo qualche altra curva e/o qualche altro punto di partenza.
   - Se abbiamo trovato $kP = O$ in qualche fase, dato che O è elemento neutro, l'iterazione non ci sposta da O, quindi dobbiamo cambiare curva e/o punto di partenza.
   - Se a un certo punto abbiamo $\text{MCD}(v, n) \neq 1$ oppure n, abbiamo trovato un fattore non banale di n.

Capitolo 12

Altri crittosistemi

Ci sono svariati altri metodi crittografici basati sui problemi intrattabili che abbiamo visto negli scorsi capitoli, o su altri problemi "difficili". In quest'ultimo capitolo ne presentiamo velocemente alcuni.

12.1 Crittosistemi di Rabin e Blum-Goldwasser

Abbiamo visto nella sezione 8.1 che conoscere le radici quadrate in $\mathbb{Z}_n$ è equivalente a conoscere la fattorizzazione di n. Il crittosistema di Rabin si basa proprio sulla difficoltà dell'estrazione della radice quadrata.

L'idea di fondo è semplice: Alice sceglie due primi p e q congrui a $3 \pmod 4$ e pubblica $n = pq$; per cifrare $m \in \mathbb{Z}_n$, Bob calcola $c := m^2 \pmod n$, e invia c ad Alice. Come abbiamo visto, l'estrazione di radice è semplice nota la fattorizzazione di n (si ricostruiscono le radici tramite il teorema cinese del resto).

Anche il crittosistema di Blum-Goldwasser si basa sul calcolo delle radici quadrate in $\mathbb{Z}_n$. Il meccanismo è semplice: ad un messaggio binario (cioè formato dalle cifre 0 e 1) viene applicata una chiave pseudo-casuale binaria, sommando bit a bit modulo 2. La chiave viene generata dal mittente a partire dalla chiave pubblica del destinatario, e il mittente invia al destinatario, oltre al messaggio cifrato, un'informazione che gli permette di ricostruire la chiave, grazie alla sua chiave privata.

1. Alice sceglie due primi p e q congrui a $3 \pmod 4$ e pubblica $n = pq$; la chiave pubblica è n, quella privata (p, q).

2. Bob vuole mandare ad Alice un messaggio $\mathbf{m} \in (\mathbb{F}_2)^N$, e deve generare quindi una chiave $\mathbf{b} \in (\mathbb{F}_2)^N$: per fare ciò, sceglie a caso $x_1 \in \mathbb{Z}_n$, $x_1 \neq 0$, e

calcola per $i = 1, \ldots, N$ la successione $x_{i+1} = x_i^2 \pmod n$. La cifra b_i della chiave è $x_i \pmod 2$.

3. Bob genera $\mathbf{c} = \mathbf{m} + \mathbf{b} \pmod 2$, ed invia ad Alice $(\mathbf{c}, x_{N+1})$.

4. Per decodificare, Alice calcola la chiave $\mathbf{b}$ estraendo in sequenza le radici a partire da x_{N+1}.

12.2 Protocollo di McEliece

Il protocollo di McEliece si basa sulla difficoltà di decodificare un generico codice lineare. L'idea è prendere un codice di cui è noto un algoritmo veloce di decodifica (per esempio, un codice di Goppa) e mascherarlo da generico codice lineare, rendendo pubblica la matrice generatrice (perturbata da un cambio di coordinate e da una permutazione).

1. Alice sceglie $G \in \mathcal{M}_{k\times n}(\mathbb{K})$, matrice generatrice per un codice t-correttore il cui algoritmo di decodifica è noto e veloce. Sceglie inoltre una matrice invertibile $S \in \mathcal{M}_k(\mathbb{K})$ e una matrice di permutazione $P \in \mathcal{M}_n(\mathbb{K})$. Calcola infine la matrice $\hat{G} = SGP$. La chiave pubblica è $(\hat{G}, t)$, mentre quella privata è (S, G, P).

2. Bob, per mandare il messaggio $\mathbf{m} \in \mathbb{K}^k$, calcola un vettore casuale $\mathbf{r} \in \mathbb{K}^n$ tale che $\mathrm{wt}(\mathbf{r}) \leqslant t$, ed invia ad Alice $\mathbf{c} = \mathbf{m}\hat{G} + \mathbf{r}$.

3. Alice, per decifrare il messaggio, calcola $\hat{\mathbf{c}} = \mathbf{c}P^{-1}$, decodifica ottenendo $\hat{\mathbf{m}}$ e recupera quindi il messaggio calcolando $\mathbf{m} = \hat{\mathbf{m}}S^{-1}$.

Perché la decifratura funziona? Notiamo che

$$\hat{\mathbf{c}} = \mathbf{c}P^{-1} = \mathbf{m}SG + \mathbf{r}P^{-1},$$

e $\mathrm{wt}(\mathbf{r}P^{-1}) = \mathrm{wt}(\mathbf{r}) \leqslant t$ dato che P è una matrice di permutazione; il processo di decodifica, quindi, elimina correttamente l'errore $\mathbf{r}P^{-1}$, e restituisce la parola del codice $\hat{\mathbf{m}} = \mathbf{m}S$. Per Alice è facile dunque recuperare il messaggio originale.

Purtroppo questo protocollo ha lo svantaggio di richiedere chiavi troppo grandi.

12.3 Protocollo *knapsack*

Il protocollo *knapsack* (zaino) si basa su una variante del *problema dello zaino*.

> *Dato uno zaino che può contenere un certo peso, e un insieme di oggetti ciascuno con il loro peso, determinare se esiste un sottoinsieme degli oggetti in grado di riempire lo zaino.*

In altre parole, dati $v_1, \dots, v_k \in \mathbb{Z}$ e $v \in \mathbb{Z}$, determinare $\boldsymbol{e} \in (\mathbb{F}_2)^k$ tale che $v = \sum_{i=1}^{k} e_i v_i$, se esiste. Per noi i v_i e v saranno numeri interi *positivi*.

Questo problema è difficile[1] nella sua generalità, ma molto semplice in casi particolari.

Definizione 12.1. Una k-upla $v_1, \dots, v_k$ è *supercrescente* se

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

per ogni $i = 1, \dots, k$. (Si intende che la somma vuota è pari a 0.)

Se la k-upla è supercrescente, è sufficiente cercare, a partire dal più grosso, il primo v_i tale che $v_i \leqslant v$, porre il rispettivo $e_i = 1$, e continuare a partire da v_{i-1} confrontando con $v - v_i$.

L'idea che sta dietro il protocollo *knapsack* è: usiamo una k-upla supercrescente come chiave privata, e una k-upla non supercrescente come chiave pubblica, ottenuta da quella privata tramite un fattore moltiplicativo e una riduzione modulo n.

1. Alice sceglie una k-upla supercrescente $(v_1, \dots, v_k)$, un intero n che sia maggiore della somma dei v_i e un intero positivo a primo con n. Alice calcola inoltre $b \equiv a^{-1} \pmod{n}$ e la k-upla $(w_1, \dots, w_k)$ dove $w_i \equiv a v_i \pmod{n}$. Alice pubblica solamente i w_i, che non sono una k-upla supercrescente.

2. Bob vuole trasmettere il messaggio $\boldsymbol{e} = (e_1, \dots, e_k)$. Per farlo, calcola $c = \sum e_i w_i$ e lo trasmette ad Alice.

3. Alice calcola
$$bC \equiv \sum_{i=1}^{k} e_i b w_i \equiv \sum_{i=1}^{k} e_i v_i \pmod{n}$$
e successivamente risolve il problema *knapsack* ricavando gli e_i.

[1] In effetti, è NP-completo.

12.4 Protocollo di Goldreich-Goldwasser-Halevi

Il protocollo di Goldreich-Goldwasser-Halevi è simile a quello di McEliece, ma è basato su reticoli anziché su codici correttori.

Definizione 12.2. Per i nostri scopi, un *reticolo* Λ è un sottogruppo di $\mathbb{Z}^n$ di rango massimo. In altre parole, dati $(\boldsymbol{v}_1, \dots, \boldsymbol{v}_n)$ vettori linearmente indipendenti in $\mathbb{Z}^n$, abbiamo

$$\Lambda = \left\{ \sum_{i=1}^{n} a_i \boldsymbol{v}_i \;\middle|\; a_1, \dots, a_n \in \mathbb{Z} \right\},$$

e quindi $\Lambda \simeq \mathbb{Z}^n$ come $\mathbb{Z}$-modulo.

Il problema su cui si basa l'algoritmo è il *closest vector problem* (CVP).

> *Data una base $\mathcal{B} = (\boldsymbol{v}_1, \dots, \boldsymbol{v}_n)$ di un reticolo Λ, e un vettore $\boldsymbol{w} \in \mathbb{Z}^n$, trovare il vettore $\boldsymbol{v} \in \Lambda$ più vicino a $\boldsymbol{w}$ (ad esempio, rispetto alla metrica euclidea su $\mathbb{Z}^n$).*

Il problema CVP è difficile se la base $\mathcal{B}$ è qualsiasi, ma diventa facilmente risolvibile per alcune basi "belle" (ad esempio, formate da vettori quasi-ortogonali). L'idea è: utilizziamo una base "bella" come chiave privata, e pubblichiamo una base qualsiasi come chiave pubblica.

1. Alice sceglie un reticolo Λ e una base "bella" $B \in \mathcal{M}_n(\mathbb{Z})$ (gli elementi della base sono le colonne della matrice B). Sceglie quindi una matrice di cambiamento di base $U \in \mathsf{GL}_n(\mathbb{Z})$, e pubblica $B' = UB$.

2. Bob vuole mandare ad Alice un messaggio $\boldsymbol{m} \in \mathbb{Z}^n$. Per fare ciò, sceglie un errore $\boldsymbol{e} \in \mathbb{Z}^n$ piccolo ed invia $\boldsymbol{c} = \boldsymbol{m}B' - \boldsymbol{e}$.

3. Per decifrare il messaggio, Alice usa la base B per risolvere il CVP e ottenere $\boldsymbol{m}B'$, da cui ricava facilmente $\boldsymbol{m}$.

12.5 NTRU

Il crittosistema NTRU è stato inventato da Hoffstein, Pipher e Silverman intorno al 1996, ed è basato anch'esso su un problema di reticoli, legato al CVP: lo *shortest vector problem* (SVP).

> *Dato un reticolo Λ, trovare il più corto vettore di Λ (ad esempio, rispetto alla norma euclidea su $\mathbb{Z}^n$).*

L'ambiente di lavoro è l'anello

$$R := \mathbb{Z}[X]/(X^n - 1),$$

cioè l'anello dei polinomi di grado al più $n-1$. Il crittosistema ha tre parametri: n, un numero primo; p, un intero piccolo (per esempio, 3); q, un intero di media grandezza (per esempio, 256). È richiesto che p e q siano coprimi.

1. Alice sceglie $n, p, q \in \mathbb{Z}$ come descritto sopra. Sceglie successivamente $f, g \in R$ con coefficienti in $\{-1, 0, 1\}$, tali che f sia invertibile modulo p e modulo q, con inversi rispettivamente f_p e f_q. La chiave pubblica è costituita dal polinomio $h \equiv g f_q \pmod{q}$.

2. Bob codifica il messaggio m come un polinomio i cui coefficienti siano compresi (supponendo p dispari) tra $-\frac{p-1}{2}$ e $\frac{p-1}{2}$, e sceglie un polinomio r a coefficienti in $\{-1, 0, 1\}$. Il messaggio codificato inviato ad Alice è $c \equiv prh + m \pmod{q}$.

3. Per decifrare il messaggio, Alice calcola $a \equiv cf \equiv (prh + m)f \equiv prg + mf \pmod{q}$ e sceglie un rappresentante con i coefficienti compresi tra $-\frac{q}{2}$ e $\frac{q}{2}$. A questo punto Alice calcola $a \pmod{p}$, ottenendo fm, e quindi recupera m moltiplicando per f_p.

La decodifica funziona perché per opportune scelte dei parametri, il polinomio $prg + mf \in R$ ha coefficienti compresi tra $-\frac{q}{2}$ e $\frac{q}{2}$, quindi non cambia se ridotto modulo q. In altre parole, quando Alice calcola $cf \pmod{q}$ e sceglie il rappresentante con coefficienti tra $-\frac{q}{2}$ e $\frac{q}{2}$, recupera *esattamente* $prg + mf \in R$. A questo punto, il resto della decodifica è facile.

Osservazione. Il legame di NTRU con i reticoli passa tramite l'isomorfismo di $\mathbb{Z}$-moduli $\mathbb{Z}[X]/(X^n - 1) \simeq \mathbb{Z}^n$. Per recuperare la chiave privata dalla chiave pubblica, un malintenzionato dovrebbe risolvere un SVP sul reticolo generato dalle righe della matrice

$$\left(\begin{array}{cccc|cccc} a & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{n-1} \\ 0 & a & \cdots & 0 & h_{n-1} & h_0 & \cdots & h_{n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a & h_1 & h_2 & \cdots & h_0 \\ \hline 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{array}\right)$$

dove gli h_i sono i coefficienti della chiave pubblica. Per ulteriori informazioni, rimandiamo all'articolo originale di Hoffstein, Pipher e Silverman.[2]

12.6 Hidden Field Equations

La famiglia di crittosistemi denominati *hidden field equations* (in breve HFE) si basa sulla difficoltà di risolvere sistemi di equazioni multivariate quadratiche. In particolare, nasconde la struttura interna di campo dell'ambiente in cui lavoriamo, mostrandolo all'esterno semplicemente come spazio vettoriale.

Ma procediamo con ordine. Sia $h(T) \in \mathbb{F}_q[T]$ un polinomio irriducibile di grado n. Come è noto, il quoziente $\mathbb{F}_q[T]/(h(T))$ è isomorfo al campo $\mathbb{F}_{q^n}$, che quindi può essere visto come $\mathbb{F}_q$-spazio vettoriale di dimensione n. Consideriamo ora un polinomio $p(X) \in \mathbb{F}_{q^n}[X]$: questo, come applicazione $\mathbb{F}_{q^n} \longrightarrow \mathbb{F}_{q^n}$, è equivalente a un vettore di n polinomi multivariati a coefficienti in $\mathbb{F}_q$. Infatti, interpretando un elemento $x \in \mathbb{F}_{q^n}$ come un vettore

$$(x_1, \ldots, x_n) \in (\mathbb{F}_q)^n,$$

è possibile scrivere $p(x)$ come $(p_1(x), \ldots, p_n(x))$, e a loro volta ciascun $p_i(x)$ può essere visto come $p_i(x_1, \ldots, x_n)$. Otteniamo quindi i polinomi $p_i(X_1, \ldots, X_n) \in \mathbb{F}_q[X_1, \ldots, X_n]$, per $i = 1, \ldots, n$.

1. Alice sceglie un polinomio irriducibile $h(T) \in \mathbb{F}_q[T]$ di grado n, quindi sceglie un polinomio $p(X) \in \mathbb{F}_{q^n}[X]$ tale che sia facilmente invertibile, come funzione $\mathbb{F}_{q^n} \longrightarrow \mathbb{F}_{q^n}$. Calcola il vettore di polinomi $\boldsymbol{v}$ associato a $T(p(S(X)))$, dove S e T sono applicazioni affini di $\mathbb{F}_{q^n}$ in sé, visto come $\mathbb{F}_q$-spazio vettoriale. La chiave pubblica è $\boldsymbol{v}$.

2. Bob, per mandare un messaggio $m \in \mathbb{F}_{q^n}$ (che interpretiamo come un elemento di $(\mathbb{F}_q)^n$), calcola $\boldsymbol{v}(m)$ e un parametro di controllo c e invia ad Alice entrambi. Il parametro di controllo è necessario perché p, come applicazione, non è iniettiva: è necessaria un'informazione extra ad Alice per capire quale controimmagine prendere.

3. Alice, per decifrare il messaggio, applica in ordine T^{-1}, p^{-1} e S^{-1} per recuperare l'originale. Il parametro di controllo è utilizzato per selezionare l'elemento nella controimmagine di p a cui applicare S^{-1} e quindi identificare il messaggio.

[2] Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman. *NTRU: A Ring Based Public Key Cryptosystem.* LNCS 1423, 1998, pagg. 267-288.

12.7 Polly Cracker

La risoluzione di un sistema polinomiale è un problema difficile. Ad esempio, mostriamo che un noto problema difficile (la 3-colorabilità di un grafo) può essere ridotta a un sistema polinomiale.

Per 3-colorabilità si intende: dato un grafo $G = (V, E)$, trovare una mappa $f : V \longrightarrow \{1, 2, 3\}$ tale che $f(v_i) \neq f(v_j)$ se $\{v_i, v_j\} \in E$. In altre parole, associare ad ogni nodo del grafo un'etichetta (ad esempio un colore: rosso, verde e blu) in modo che due nodi adiacenti abbiano colori diversi. Questo problema si può tradurre nel seguente modo: se $\#V = n$, consideriamo l'insieme di $3n$ indeterminate $\{X_1^{(1)}, \ldots, X_n^{(1)} X_1^{(2)}, \ldots, X_n^{(2)}, X_1^{(3)}, \ldots, X_n^{(3)}\}$, e consideriamo lo spazio dei polinomi in tali indeterminate, a coefficienti in $\mathbb{F}_2$. Se interpretiamo $X_i^{(k)} = 1$ come "al vertice v_i è assegnato il colore k", allora per trovare una 3-colorazione è sufficiente risolvere il sistema

$$\begin{cases} (X_i^{(k)})^2 - X_i^{(k)} = 0 \quad \forall\, i = 1, \ldots, n \;\forall\, k = 1, 2, 3 & (1) \\ X_i^{(1)} X_i^{(2)} = 0,\ X_i^{(2)} X_i^{(3)} = 0,\ X_i^{(3)} X_i^{(1)} = 0 \quad \forall\, i = 1, \ldots, n & (2) \\ X_i^{(1)} + X_i^{(2)} + X_i^{(3)} = 1 \quad \forall\, i = 1, \ldots, n & (3) \\ X_i^{(k)} X_j^{(k)} = 0 \quad \forall\, \{v_i, v_j\} \in E \;\forall\, k = 1, 2, 3 & (4) \end{cases}$$

che in termini di colorazione si traduce con

(1) la soluzione può essere solo 0 oppure 1;

(2) ogni vertice ha un solo colore;

(3) ogni vertice è colorato;

(4) vertici adiacenti hanno colori diversi.

Il crittosistema Polly Cracker si basa appunto sulla difficoltà di risolvere sistemi polinomiali e di calcolare basi di Gröbner di ideali polinomiali. Nel seguito, per brevità di scrittura, indicheremo con $\mathbf{X}$ l'insieme delle indeterminate $X_1, \ldots, X_n$.

1. Alice sceglie una base di Gröbner $\mathcal{G}$ di un ideale $I \subseteq \mathbb{K}[\mathbf{X}]$. La base $\mathcal{G}$ è mantenuta privata, mentre è pubblico l'insieme dei rappresentanti canonici di $\mathbb{K}[\mathbf{X}]/I$, che rappresenta lo spazio dei messaggi in chiaro.

2. Alice sceglie un insieme $\mathcal{B} = \{q_j\}$ di polinomi in I, ad esempio prendendo un polinomio a caso $\tilde{q}_j$ e calcolando $q_j := \tilde{q}_j - \overline{q}_j$, dove $\overline{q}_j$ è il resto della divisione di $\tilde{q}_j$ per $\mathcal{G}$; l'insieme $\mathcal{B}$ è la chiave pubblica di Alice.

3. Bob, per mandare un messaggio $m \in \mathbb{K}[\mathbf{X}]/I$ ad Alice, sceglie $p \in (\mathcal{B})$, cioè un polinomio nell'ideale generato da $\mathcal{B}$, e invia $c = m + p$.

4. Alice deve solo ridurre c con la base di Gröbner per recuperare il messaggio in chiaro.

Purtroppo questo sistema presenta alcuni inconvenienti.

- Le basi di Gröbner si calcolano con un costo esponenziale nel numero di indeterminate, ma tale costo non è mai doppiamente esponenziale (tranne in casi costruiti *ad hoc*).

- Potrebbe non essere necessario calcolare l'intera base di Gröbner, ma limitarne il grado: i polinomi di grado elevato in $\mathcal{G}$ intervengono solo se c contiene elementi di grado elevato. Per evitare ciò, potremmo usare polinomi di grado elevato per mascherare il messaggio, ma questo alza notevolmente il costo della cifratura, che diventa paragonabile a quello della decifratura.

- Potremmo pensare di usare polinomi sparsi per non appesantire il calcolo, ma così facendo daremmo informazioni sul *pattern* dei monomi utilizzati. Infatti ci sono pochi polinomi in I sparsi; analizzando le differenze tra i monomi di c, è facile costruire un polinomio che sta in I, e utilizzarlo per eliminare una parte della maschera.

12.8 Protocolli a chiave simmetrica: DES e AES

Presentiamo qui un paio di algoritmi di cifratura a chiave *simmetrica*: la stessa chiave è usata per cifrare e decifrare. Questi protocolli vengono utilizzati perché normalmente i protocolli a chiave asimmetrica richiedono molto costo computazionale su messaggi lunghi, quindi si usa la cifratura asimmetrica per trasmettere la chiave di una cifratura simmetrica, con cui poi si cifra il messaggio vero e proprio.

Il primo protocollo che consideriamo è il *Data Encryption Standard*, o DES. Esso è stato scelto come standard dal governo statunitense a cavallo degli anni '70, ed abbandonato all'inizio del nuovo millennio perché la maggiore potenza di calcolo lo ha reso violabile tramite un attacco di forza bruta.

Il messaggio da cifrare viene suddiviso in blocchi da 64 bit, e DES si applica a ciascun blocco. La chiave del DES è una sequenza di 64 bit, di cui 56 effettivamente usati per cifrare e 8 di controllo. La cifratura avviene per stadi successivi (*round*), uguali tra loro ma eseguiti con chiavi diverse (*chiavi di round*,

tutte derivate dalla chiave iniziale). Prima e dopo l'intero processo, vengono applicate al blocco due permutazioni, una inversa dell'altra, che non hanno influenza sulla cifratura.

Un blocco da 64 bit viene spezzato in due parti da 32 bit, e i *round* agiscono alternativamente su una parte e sull'altra; in altre parole, se $F : (\mathbb{F}_2)^{32} \longrightarrow (\mathbb{F}_2)^{32}$ è la funzione che viene applicata al blocco da 32 bit, il messaggio (A, B) (con A e B di 32 bit), dopo un *round*, viene trasformato nel messaggio $(B, A \oplus F(B))$, dove $\oplus$ indica la somma modulo 2. Vediamo in dettaglio come agisce F.

1. Il blocco da 32 bit viene espanso a 48 bit, duplicandone alcuni. (L'algoritmo di espansione è pubblico.)

2. Al blocco di 48 bit viene sommata (modulo 2) una chiave di *round* estratta dalla chiave globale. (Anche qui, il processo di estrazione della chiave è pubblico.)

3. Il blocco da 48 bit viene diviso in 8 parti di 6 bit, ciascuna processata da una *S-box*: una tabella che associa a ogni sequenza di bit un'altra sequenza da 4 bit. (La struttura della S-box è pubblica.) Le parti vengono dunque ricomposte, formando un blocco da 32 bit.

4. Infine, il blocco passa attraverso una *P-box*, che ha il compito di permutarne i bit. (Anche in questo caso, la struttura della P-box è pubblica.)

In totale vengono eseguiti 16 *round*.

Alcuni modi per attaccare DES sono il *known message attack*, in cui si conoscono alcune coppie messaggio in chiaro-messaggio cifrato, e da queste si cerca di recuperare la struttura della chiave; oppure il *chosen message attack*, in cui si ha accesso a una macchina che cifra con la chiave che vogliamo scoprire, e scegliamo il messaggio da cifrare in modo che la cifratura riveli alcune proprietà della chiave.

Con l'aumentare della potenza di calcolo delle macchine, il DES venne potenziato nel cosiddetto *Triple DES*: DES ripetuto tre volte, con delle varianti in base alle chiavi usate. Si può scegliere infatti di applicare tre volte la stessa chiave, scegliere tre chiavi diverse, o anche alternare cifratura e decifratura (per esempio, si può applicare k_1, poi k_2, e infine k_1^{-1}).

Nel 1998, è stato aperto un bando di concorso per cercare il successore del DES. Il vincitore è stato Rijndael (dal nome dei progettisti Vincent Rijmen e Joan Daemen), poi ribattezzato AES (*Advanced Encryption Standard*).

AES agisce su blocchi di messaggio lunghi 128 bit, e può avere chiavi lunghe 128, 192 o 256 bit. Il blocco da 128 bit è interpretato come blocco da 16 byte

disposti in una matrice 4×4, dove un byte è visto come elemento di

$$\mathbb{F}_{256} \simeq \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1) \cdot$$

Anche AES agisce per *round*, e precisamente vengono eseguiti 10, 12 oppure 14 *round* in base alla lunghezza della chiave. Vediamo più in dettaglio.

1. La chiave viene usata per generare le chiavi di *round*.
2. Primo *round*: si somma (modulo 2) la chiave di *round* e non si fa altro.
3. *Round*:
   (a) si applica una S-box, che rappresenta una piccola variazione della mappa $x \longmapsto x^{-1}$ in $\mathbb{F}_{256}$: questa mappa ha buone proprietà non lineari che la rendono adatta alla cifratura;
   (b) si *shiftano* le righe (precisamente, la i-esima di $i - 1$ posizioni verso sinistra);
   (c) si mischiano le colonne, operazione che può essere compiuta tramite moltiplicazione per un polinomio fissato (e riduzione seguente modulo $X^8 + X^4 + X^3 + X + 1$);
   (d) si somma (modulo 2) la chiave di round.
4. Ultimo *round*: come un *round* normale, ma non vengono mischiate le colonne.

Tutte le operazioni sono lineari, a parte la S-box: senza di essa, AES sarebbe facilmente attaccabile. In effetti, tutte le operazioni sono polinomiali, ma i problemi da risolvere per un eventuale attacco sono intrattabili.

Versioni ridotte di AES sono state rotte, mentre la versione completa resta forte, tanto da essere autorizzata a cifrare informazioni di livello SECRET e TOP-SECRET.

12.9 Identificazione: FS e FFS

Terminiamo la trattazione con il problema dell'identificazione, o dell'autenticazione. Come può una persona provare la propria identità?

Uno dei metodi più intuitivi è inviare la propria password. Questo però è estremamente fragile: il ricevente deve conservare una copia della password (per il confronto), e quindi può essere attaccato; oppure un malintenzionato può intercettare la password e usarla per identificarsi al posto del trasmittente.

Si cercano dunque *zero-knowledge protocols*: chi si vuole identificare dimostra di possedere un'informazione senza dare informazioni su di essa.

Il protocollo di Fiat-Shamir (FS) si basa sulla difficoltà di calcolare radici quadrate in $\mathbb{Z}_n$, con $n = pq$, senza conoscere la fattorizzazione di n. Nella sua descrizione, introdurremo due personaggi differenti rispetto ad Alice e Bob: Peggy (la *prover*, cioè quella che vuole dimostrare la sua identità) e Victor (il *verifier*, cioè colui che vuole verificare l'identità di Peggy).

1. Un'*autorità di certificazione* sceglie due fattori primi p, q, e rende pubblico il loro prodotto n.
2. Peggy sceglie un numero segreto s, primo con n, e pubblica $v \equiv s^2 \pmod{n}$, dichiarando "conosco la radice quadrata di v".
3. Quando Victor richiede a Peggy di autenticarsi, quest'ultima sceglie un numero r a caso, e manda $x \equiv r^2 \pmod{n}$ a Victor.
4. Victor manda un bit $e \in \{0, 1\}$ a Peggy.
5. Peggy risponde con $a \equiv rs^e \pmod{n}$.
6. Victor verifica che $a^2 \equiv xv^e \pmod{n}$.

Mandando e, Victor sceglie di verificare una delle informazioni che solo Peggy possiede (r e rs). Un malintenzionato (Eva) che vuole farsi passare per Peggy deve tirare a indovinare e: ha quindi probabilità $1/2$ di identificarsi come Peggy. Infatti, Eva sceglie y a caso, cerca di indovinare quale e sceglierà Victor e calcola $x \equiv y^2 v^{-e} \pmod{n}$, che invia a Victor. Quando quest'ultimo invia il suo e, Eva risponde con y. Se Eva ha indovinato e, passa il test; altrimenti, viene beccata. Si può ripetere il test m volte per abbassare la possibilità che Eva passi il test a $(1/2)^m$.

Questo protocollo non è *zero-knowledge*: è rivelata infatti l'esistenza di un quadrato (x). Per ovviare a questo problema, si passa all'algoritmo Feige-Fiat-Shamir (FFS): viene introdotto un bit di segno, che impedisca di riconoscere i quadrati. Inoltre (ma questo si poteva fare anche per FS) vengono scelti k numeri segreti $s_1, \ldots, s_k$ di cui calcolare i quadrati $v_1, \ldots, v_k$.

1. Peggy sceglie $s_1, \ldots, s_k$, con $\text{MCD}(s_i, n) = 1$, e rende pubblici $v_i \equiv s_i^2 \pmod{n}$.
2. Su richiesta di Victor, Peggy sceglie r a caso, un segno $s \in \{\pm 1\}$, e invia $x \equiv sr^2 \pmod{n}$.

3. Victor manda k bit $(e_1, \ldots, e_k) \in \{0,1\}^k$.
4. Peggy calcola $y \equiv r s_1^{e_1} \cdots s_k^{e_k} \pmod{n}$ e lo invia a Victor.
5. Victor verifica che $y^2 \equiv \pm x v_1^{e_1} \cdots v_k^{e_k}$.

Anche in questo caso, Eva ha $(1/2)^k$ di passare il test (deve indovinare l'intera sequenza di bit $(e_1, \ldots, e_k)$), che può essere ulteriormente abbassata a $(1/2)^{km}$ se il test è ripetuto m volte.