Appunti di Teoria di Codici e Crittografia
Prof. Traverso

Carlo Sircana, Giovanni Barbarino, Riccardo Morandin

Indice

Capitolo 1

Codici correttori

In questa prima parte, studieremo i codici e il loro utilizzo per la correzione degli errori nella trasmissione di messaggi.

Informalmente, quando si manda un messaggio tramite una linea digitale, una delle operazioni che il calcolatore compie automaticamente è quella di scegliere un *dizionario* o *codice*, ossia un insieme di stringhe binarie rappresentanti tutti i possibili messaggi. Dopo che il messaggio viene trasformato in una stringa del codice (*codifica* del messaggio), questo viene trasmesso e in tale passaggio alcuni bit del messaggio possono essere modificati per cause esterne. Il ricevente deve essere in grado di *correggere* l'errore, ricavare la stringa di codice spedita, e da questa ricostrure il messaggio originale (operazione detta *decodifica* del messaggio). Quello su cui ci concentreremo, sarà studiare particolari tipi di codici, prestando particolare attenzione all'operazione di correzione del messaggio.

Consideriamo allora un alfabeto finito, che modellizziamo tramite un campo finito K. Un elemento del codice è una stringa finita di elementi di K, ossia un elemento di K^n. A noi interessano però solo particolari tipi di codici:

Definizione 1.1. Un codice lineare è un sottospazio vettoriale dello spazio delle stringhe K^n, dove K è un campo finito.

La trasmissione è allora una applicazione $K^n \to K^n$. Il problema che ci poniamo è quello di correggere gli errori di trasmissione: infatti non è detto che l'immagine del codice C tramite l'applicazione di trasmissione φ sia ancora contenuta in C.

La strategia che utilizzeremo sempre è di tipo probabilistico: dato $v \in C$ e presa $\varphi(v)$ la sua immagine tramite la trasmissione, per ricostruire il messaggio iniziale sceglieremo l'elemento del codice più vicino a $\varphi(v)$ secondo una opportuna distanza. Per questa ragione, supporremo che la probabilità che un singolo elemento della stringa sia sbagliato sia minore di $1/2$, e che gli errori siano indipendenti carattere per carattere. La distanza che consideriamo è la seguente:

Definizione 1.2 (Distanza di Hamming)**.** Sia K un campo finito e siano $v, w \in K^n$. Chiamiamo distanza di Hamming tra v, w

$$h(v, w) = \#\{\text{componenti non nulle di } v - w\}$$

Indicheremo con $h(v) = \#\{v_i \neq 0\}$ il peso di Hamming[1].

[1]Non si tratta in generale di una norma, tranne nel caso $K = \mathbb{F}_2$.

La distanza di Hamming è effettivamente una metrica, in quanto:

- $h(v,w) = h(w,v)$ per ogni $v, w \in K^n$
- $h(v,w) \geq 0$ per ogni $v, w \in K^n$
- $h(v,w) = 0$ se e solo se $v = w$
- $h(v,w) \leq h(v,z) + h(z,w)$ per ogni $v, w, z \in K^n$

Si osserva inoltre che la distanza di Hamming soddisfa le seguenti proprietà:

- $h(v, w+z) = h(v-z, w)$ per ogni $v, w, z \in K^n$ (invarianza per traslazione);
- $h(av, aw) = h(v,w)$ per ogni $v, w \in K^n$ e $a \in K^*$ (invarianza per scalare).

Dunque quello che vogliamo fare è applicare la strategia di Maximum Likelihood Decoding rispetto alla distanza di Hamming: la speranza è che il numero di bit alterati durante la trasmissione sia piccolo. In generale, questo è un problema difficile in quanto NP-completo.

Rappresentazione di codici Occupiamoci ora di capire come rappresentare un codice. Le scelte sono chiaramente due: possiamo dare una base del codice come sottospazio di K^n, oppure darne delle equazioni cartesiane.

Definizione 1.3. Una matrice generatrice di un codice C è una matrice le cui righe sono un insieme di generatori per il codice. Una matrice di parità di un codice C è una matrice le cui colonne sono delle equazioni che identificano C.

Nessuna delle due rappresentazioni è preferibile; a seconda del codice preso in considerazione sarà più semplice trovare una o l'altra presentazione. Esiste un metodo particolarmente semplice per passare da una rappresentazione all'altra. Supponiamo che le righe di una matrice generatrice siano linearmente indipendenti, ossia che i vettori siano una base del codice. Possiamo ridurre la matrice in forma di Gauss, ottenendo

$$\left(\begin{array}{ccc|c} 1 & & & \\ & \ddots & & B \\ & & 1 & \end{array}\right)$$

Allora la matrice

$$\left(\begin{array}{c} -B \\ \hline \begin{array}{ccc} 1 & & \\ & \ddots & \\ & & 1 \end{array} \end{array}\right)$$

è una matrice di parità del codice. Infatti ogni colonna di quest'ultima rappresenta un'equazione che si annulla su ogni riga della matrice generatrice. Chiaramente si può fare anche il viceversa, cioè portarsi da una matrice di parità a una generatrice, purché le equazioni siano indipendenti (possiamo sempre ridurci a questo caso).

Esempio (Codici di ripetizione). Una prima idea per facilitare la correzione è quella di mandare il messaggio più volte. Supponiamo per esempio di voler mandare la stringa $1 \in K$. Allora, per essere quasi certi di saper ricostruire il messaggio, potremmo mandare $v = (1, \dots, 1) \in K^n$ e ricostruire il messaggio per maggioranza, ossia decretare che il messaggio iniziale era l'elemento di K che ricorre più volte in $\varphi(v)$. Chiaramente è conveniente che n sia dispari; se infatti n è pari non è detto che la decisione sia univoca, ossia potrebbero esistere messaggi ugualmente probabili.

Esempio (Codice di parità). Supponiamo di avere $K = \mathbb{F}_2$ e prendiamo come codice il sottospazio $C \subseteq (\mathbb{F}_2)^8$ formato da tutti i vettori la cui somma delle componenti è 0. Questo codice è facilmente descrivibile tramite la matrice di parità, in quanto C è un sottospazio di dimensione 7 descritto dall'equazione $\sum_{i=1}^{8} x_i = 0$. Dunque la matrice di parità è

$$\begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

Questo codice riconosce se c'è un solo errore, ma non sa distinguere tra 2 errori.

Esempio (Codice di Hamming). Consideriamo $K = \mathbb{F}_2$ e il codice $C \subseteq (\mathbb{F}_2)^7$ avente come matrice di parità

$$A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

ossia nella riga i-esima contiene la scrittura di i in base 2. Dato che le colonne sono linearmente indipendenti, il codice è un sottospazio di dimensione 4. Con questo codice, trovare un errore è molto semplice, supponendo che ce ne sia uno solo. Sia $v \in \mathbb{F}_2^8$ e consideriamo il vettore $v \cdot A$; questo fornisce i coefficienti della scrittura in base due della posizione nel quale si trova l'errore. Per esempio, consideriamo il vettore $v = (1, 0, 1, 1, 0, 0, 1)$. Allora $v \cdot A = (0, 0, 1)$ e dunque individuiamo un probabile errore in posizione 1.

Quanto visto nell'esempio suggerisce la seguente definizione:

Definizione 1.4. Sia C un codice con matrice di parità A e sia $v \in K^n$. Chiamiamo sindrome il vettore $v \cdot A$.

La sindrome è nulla se $v \in C$, altrimenti può fornire informazioni sulla posizione degli errori. Nel caso del codice di Hamming, la sindrome individua un probabile errore e siamo sempre in grado di ricostruire un elemento del codice da cui molto probabilmente proveniva il messaggio ricevuto. Anche il codice di ripetizione nel caso n dispari è sempre in grado di ricostruire un probabile messaggio iniziale. Codici di questo tipo vengono detti perfetti; vedremo a breve una definizione formale.

Lunghezza, Dimensione, Distanza

Definizione 1.5. Sia $C \subseteq K^n$ un codice. Definiamo

- la lunghezza di C come la dimensione n dello spazio K^n in cui è contenuto.
- la dimensione di C come la dimensione di C come sottospazio di K^n.
- la distanza di C come $\min\{h(c) \mid c \in C \setminus \{0\}\}$, dove h è il peso di Hamming.

La distanza di un codice indica quanti errori può correggere. Idealmente, fissata la lunghezza di un codice, siamo interessati a trovare un codice con dimensione e distanza più grandi il possibile. Un codice può correggere senza ambiguità fino a $\lfloor (d-1)/2 \rfloor$ errori. Infatti, se un elemento $v \in K^n$ ha meno di questi errori, esiste un unico elemento del codice di distanza minima da v.

Esempio (Distanza del codice di Hamming). Il codice di Hamming 1 ha distanza 3. Infatti l'elemento $v = (0,0,1,1,0,0,1)$ appartiene al codice e $h(v) = 3$, da cui segue che $d \leq 3$. Supponiamo per assurdo che esista un elemento del codice di distanza 2. Questo significa che esistono due righe della matrice di parità linearmente dipendenti e questo è chiaramente assurdo. Dato che nessuna riga della matrice è nulla e dunque non esistono vettori di peso 1, si ha che la distanza del codice di Hamming è 3 e dunque corregge un solo errore.

Il ragionamento effettuato nel caso del codice di Hamming ci porta a una semplice disuguaglianza che lega lunghezza, dimensione e distanza. Infatti la distanza di un codice è uguale al minimo numero di righe linearmente dipendenti della matrice di parità, quindi è sempre minore o uguale al suo rango più 1. D'altronde il rango della matrice di parità è uguale a $n - m$, da cui

Teorema 1.6 (Disuguaglianza di Singleton)**.** Sia C un codice di lunghezza n, dimensione m e distanza d. Allora $n - m \geq d - 1$.

Definizione 1.7. Un codice C che soddisfa l'uguaglianza nella disuguaglianza di Singleton viene detto MDS ("Maximum Distance Separable").

Un'altra nota disuguaglianza tra le misure di un codice è la seguente:

Teorema 1.8 (Disuguaglianza di Hamming)**.** Sia C un codice di lunghezza n, dimensione m e distanza d su $K = \mathbb{F}_q$ e sia $t = \lfloor (d-1)/2 \rfloor$. Allora vale

$$q^m \sum_{i=0}^{t} \binom{n}{i} (q-1)^i \leq q^n$$

Dimostrazione. Sappiamo che, comunque presi due elementi del codice v, w, $B(v,t) \cap B(w,t) = \emptyset$ (dove $B(x,t)$ indica la palla chiusa di centro x e raggio t, rispetto alla distanza di Hamming). Di conseguenza la somma delle cardinalità di $B(v,t)$ al variare di $v \in C$ è minore di q^n, il numero di elementi di K^n. Il numero s_i di elementi a distanza i da un elemento del codice è uguale a

$$s_i = \binom{n}{i} (q-1)^i$$

da cui la tesi. □

Notiamo che se vale l'uguaglianza ogni elemento di K^n è a distanza $\leq t$ da un elemento del codice e dunque può essere corretto senza ambiguità.

Definizione 1.9. Un codice per il quale vale l'uguaglianza nella disuguaglianza di Hamming si dice codice perfetto.

L'ultima disuguaglianza che studiamo è la disuguaglianza di Gilbert-Varshamov:

Teorema 1.10 (Disuguaglianza di Gilbert-Varshamov)**.** Sia $A(n, d)$ la massima dimensione che può avere un codice di lunghezza n e distanza d. Allora

$$q^{A(n,d)} \sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i \geq q^n$$

Dimostrazione. Dimostriamo che se un codice C di misure n, m, d non soddisfa la disuguaglianza, allora esiste $C' \supseteq C$ di misure $n, m+1, d$. Se

$$q^m \sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i < q^n$$

allora esiste un elemento $x \in K^n$ tale che $\min\{h(c, x) \mid c \in C\} \geq d$. Il codice $C + \langle x \rangle$ ha distanza d; sia infatti $y = ax + w$, con $w \in C$ e $a \in K$. Dunque, sfruttando l'invarianza per traslazioni e per moltiplicazione per scalare della distanza di Hamming,

$$h(ax + w, 0) = h(x + c^{-1}w) = h(x, -a^{-1}w) \geq d$$

dove l'ultima disuguaglianza utilizza il fatto che $-a^{-1}w \in C$. □

La disuguaglianza di Gilbert-Varshamov è molto importante; idealmente infatti un codice dovrebbe avere, a lunghezza n fissata, dimensione e distanza più grandi il possibile, in modo da dover trasmettere pochi bit senza informazioni e saper correggere il maggior numero di errori. La dimostrazione di questa disuguaglianza è costruttiva e dunque in linea di principio permette, preso un codice C, di migliorarne le misure. Nella pratica, questo non è realizzabile, perché per un codice qualunque la correzione del messaggio è un problema NP-completo e spesso, dopo aver applicato la costruzione illustrata nella dimostrazione, perdiamo delle buone proprietà del codice che permettevano una facile codifica. Noi studieremo dei codici per i quali la correzione è semplice; purtroppo non sempre questi saranno ottimali rispetto a tale disuguaglianza.

Per modificare le misure di un codice e renderlo più efficente, si possono eseguire alcune operazioni:

Somma Dati due codici $C_1 \subseteq K^{n_1}$ e $C_2 \subseteq K^{n_2}$, possiamo definire la somma diretta come il codice $C_1 \oplus C_2 \subseteq K^{n_1+n_2}$. In questo caso, la lunghezza è la somma delle lunghezze, la dimensione è la somma delle dimensioni e la distanza il minimo delle distanze.

Estensione e restrizione Dato K un campo e preso L una sua estensione, possiamo estendere il codice mediante il prodotto tensore. Dato allora un codice $C \subseteq K^n$, otteniamo tramite prodotto tensore il codice $C \otimes_K L \subseteq L^n$, che ha lunghezza e dimensione uguale alla lunghezza di C e distanza minore o uguale a quello di C. Dato invece un codice $C' \subseteq L^n$, possiamo considerare il codice $C = C \cap K^n$. In questo caso, la lunghezza rimane invariata, la dimensione può decrescere e la distanza può aumentare.

Pizzicare Sia $C \subseteq K^n$ un codice e supponiamo che per ogni $c \in C$ la i-esima componente c_i sia nulla. Allora possiamo ignorare la i-esima componente, ottenendo un codice $C' \subseteq K^{n-1}$ di dimensione e distanza uguale ma lunghezza uno in meno.

1.1 Codici Ciclici e BCH

Definizione 1.11. Sia $C \subseteq K^n$ un codice lineare. Diciamo che C è ciclico se, comunque preso un elemento $c = (c_0, \dots, c_{n-1}) \in C$, ogni sua permutazione ciclica $(c_i, c_{i+1}, \dots, c_{n-1}, c_0, \dots, c_{i-1})$ appartiene al codice.

Un codice ciclico ammette una rappresentazione come polinomio. Al vettore $(c_0, \dots, c_{n-1})$ possiamo associare il polinomio $\sum_{i=0}^{n-1} c_i x^i$. L'invarianza per l'azione di una permutazione ciclica possiamo tradurla con l'azione di x modulo $x^n - 1$ e questo fornisce al codice una struttura di ideale in

$$A = {}^{K[x]}\!\diagup\!_{(x^n - 1)}$$

Infatti, la chiusura per permutazione ciclica impone che il codice sia chiuso per la moltiplicazione per x ; inoltre il codice è chiuso per somma e dunque deve essere un sottomodulo di A, ossia un ideale.

Consideriamo la proiezione $K[x] \to A$; all'ideale I individuato dal codice corrisponde un solo ideale J di $K[x]$ contenente $x^n - 1$; dato che $K[x]$ è un PID, esiste un unico polinomio monico $\varphi \in K[x]$ che genera J, che chiamiamo polinomio generatore del codice. Notiamo che φ genera come ideale sia I che J, dunque, a meno di moltiplicazione per costante, è il polinomio di grado minore contenuto nel codice. Se lo scriviamo come $\varphi(x) = \sum_{i=0}^{r} c_i x^i$, allora avremo $r < n$, e per ogni polinomio $p(x)$ del codice, avremo $p(x) = \varphi(x)r(x)$, con $r(x)$ di grado $deg(p) - deg(\varphi) < n - r$. Ciò vuol dire che il codice sarà generato come spazio vettoriale su K dai polinomi

$$\varphi(x) \quad x\varphi(x) \quad x^2\varphi(x) \quad \dots \quad x^{n-r-1}\varphi(x)$$

In base a questo, una matrice generatrice del codice è la seguente:

$$\begin{pmatrix} c_0 & c_1 & \cdots & c_r & 0 & \cdots & 0 \\ 0 & c_0 & \cdots & c_{r-1} & c_r & \cdots & 0 \\ \vdots & \vdots & \ddots & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & c_0 & \cdots & c_{r-1} & c_r \end{pmatrix}$$

In particolare, a meno che il codice non contenga solo il polinomio costantemente nullo, la matrice generatrice ha rango massimo, e dunque il codice ha dimensione $n - r$ e lunghezza n, mentre sappiamo solo che la distanza è minore o uguale ad $r + 1$ perché $\varphi(x)$ sta nel codice, ed ha al massimo $r + 1$ coefficienti non nulli (o anche, alternativamente, per la disuguaglianza di Singleton).

Esempio. Consideriamo p primo, $n \geq 1$ e sia $q = p^n$. Allora $\mathbb{F}_q^*$ è ciclico e possiamo sceglierne un generatore $\alpha \in \mathbb{F}_q^*$. Il polinomio $f = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^r)$ con $r \leq q - 1$ genera allora un codice ciclico; studiamo le misure di questo codice. Dato che che $f \mid x^{q-1} - 1$ possiamo pensare il codice come un ideale di $\mathbb{F}_q[x]/(x^{q-1} - 1)$. Dunque la lunghezza del codice è $q - 1$ e la sua

dimensione è $m = (q-1) - r$. Per studiarne la distanza, scriviamo una matrice di parità del codice. Un polinomio appartiene al codice ciclico se e solo se appartiene all'ideale generato da f. Dunque, affinché un polinomio appartenga al codice, dovrà annullarsi in α e nelle sue potenze, e una matrice di parità sarà

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha & \alpha^2 & \cdots & \alpha^r \\ \vdots & \vdots & & \vdots \\ \alpha^{q-2} & \alpha^{2(q-2)} & \cdots & \alpha^{r(q-2)} \end{pmatrix}$$

Studiamo il rango dei minori quadrati: scelti degli indici $i_1 < \cdots < i_r$ e isolata la sottomatrice

$$A = \begin{pmatrix} \alpha^{i_1} & \alpha^{2i_1} & \cdots & \alpha^{ri_1} \\ \vdots & \vdots & & \vdots \\ \alpha^{i_r} & \alpha^{2i_r} & \cdots & \alpha^{ri_r} \end{pmatrix},$$

vogliamo mostrare che è invertibile; per farlo, mostriamo che il determinante è non nullo. Per multilinearità, raccogliendo dalla riga j-esima α^{i_j}, otteniamo

$$\det(A) = \alpha^{i_1} \dots \alpha^{i_r} \det \begin{pmatrix} 1 & \alpha^{i_1} & \cdots & \alpha^{(r-1)i_1} \\ 1 & \alpha^{i_2} & \cdots & \alpha^{(r-1)i_2} \\ \vdots & \vdots & & \vdots \\ 1 & \alpha^{i_r} & \cdots & \alpha^{(r-1)i_r} \end{pmatrix}$$

Questa è una matrice di Vandermonde e $\alpha^{i_j} \neq \alpha^{i_k}$ per $j \neq k$, quindi A è invertibile. Di conseguenza la distanza di questo codice è almeno $r+1$, ma per quanto visto sopra, la distanza deve essere esattamente $r+1$.

Un codice come quello dell'esempio è detto codice di Reed-Solomon:

Definizione 1.12. Un codice di Reed-Solomon $R(d, n)$ è un codice ciclico su $\mathbb{F}_q$ il cui polinomio generatore è

$$f(x) = (x-\alpha)\cdots(x-\alpha^{d-1}) \in {}^{\mathbb{F}_q[x]}\!\diagup\!_{(x^n-1)}$$

e α è un elemento di ordine $q-1$ in $\mathbb{F}_q$.

Più in generale, possiamo definire i codici BCH:

Definizione 1.13. Un codice BCH di distanza designata d di lunghezza n su $\mathbb{F}_q$ è un codice ciclico C il cui polinomio generatore $g \in \mathbb{F}_q[x]$ è il minimo comune multiplo dei polinomi minimi di $\beta^l, \dots \beta^{l+d-2}$, dove β è una radice n-esima dell'unità. Se $l = 1$, C si dice BCH in senso stretto.

Esempio. Su $\mathbb{F}_q$, vogliamo costruire un codice BCH e lunghezza n e distanza designata d. Consideriamo una radice primitiva n-esima dell'unità $\alpha \in \overline{\mathbb{F}}_q$ con $(n, q) = 1$, e il polinomio

$$f(x) = (x-\alpha^s)(x-\alpha^{s+1})\cdots(x-\alpha^{s+d-2})$$

Come visto nell'esempio precedente, questo genera un codice di distanza d su $\mathbb{F}_{q^r}$(il più piccolo campo che contiene α), e se $\varphi_s, \dots, \varphi_{s+d-2}$ sono i polinomi

minimi di $\alpha^s, \ldots, \alpha^{s+d-2}$ su $\mathbb{F}_q$, allora il codice BCH corrispondente è il codice ciclico generato da

$$g = \text{lcm}(\varphi_s, \ldots, \varphi_{s+d-2})$$

Per motivare questo, basta notare che la restrizione di un codice ciclico é ancora ciclico, dunque generato da un polinomio, e questo deve essere il polinomio di grado minore che ha per radici $\alpha^s, \ldots, \alpha^{s+d-2}$, ossia esattamente g. Notiamo che in questo modo la lunghezza del codice resta invariata, mentre la dimensione e la distanza cambiano. In particolare, la dimensione decresce e la distanza cresce. Per questo motivo, nella definizione dei codici BCH, si parla di distanza designata invece che solo di distanza.

Occupiamoci ora della decodifica di un codice BCH. Sia quindi C un codice BCH in senso stretto con elemento primitivo α di lunghezza n e distanza designata $d = 2t + 1$ (la decodifica di un BCH generale si fa in modo analogo). Consideriamo quindi una parola del codice $a(x)$ e supponiamo di ricevere, dopo la trasmissione, la parola $b(x)$. Consideriamo il polinomio d'errore

$$e(x) = b(x) - a(x) = \sum_{i=0}^{n-1} e_i x^i$$

L'obiettivo è quello di individuare la posizione degli errori e i valori $e_i \neq 0$, nel'ipotesi che il numero di errori non superi t. Trovare direttamente il polinomio $e(x)$ è difficile, perché ha grado molto alto. L'idea è allora quella di collegare ad $e(x)$ altri due polinomi. Sia M l'insieme degli indici tali che $e_i \neq 0$. Definiamo allora il polinomio locatore dell'errore come

$$\sigma(x) = \prod_{i \in M} (1 - \alpha^i x)$$

Questo polinomio individua la posizione di un errore: infatti $e_i \neq 0$ se e solo se $\sigma(\alpha^{-i}) = 0$. Conoscere gli zeri di questo polinomio permette quindi di conoscere la posizione degli errori. Introduciamo poi il polinomio valutatore dell'errore

$$\omega(x) = \sum_{i \in M} e_i \alpha^i x \prod_{\substack{j \neq i \\ j \in M}} (1 - \alpha^j x)$$

Se conoscessimo entrambi questi polinomi, avremmo individuato e, in quanto sapremmo le posizioni degli errori e

$$\omega(\alpha^{-i}) = e_i \prod_{\substack{j \neq i \\ j \in M}} (1 - \alpha^{j-i}) = -e_i \frac{\sigma'(\alpha^{-i})}{\alpha^i}$$

Cerchiamo dunque un metodo per trovare questi due polinomi. Notiamo che

$$\begin{aligned}\frac{\omega(x)}{\sigma(x)} &= \sum_{i\in M} e_i \frac{\alpha^i x}{1-\alpha^i x} \\ &= \sum_{i\in M} e_i \sum_{k=1}^{\infty} (\alpha^i x)^k \\ &= \sum_{k=1}^{\infty} x^k \sum_{i\in M} e_i \alpha^{ik} \\ &= \sum_{k=1}^{\infty} x^k e(\alpha^k)\end{aligned}$$

Dunque sappiamo esprimere il quoziente ω/σ come serie formale. Inoltre, conosciamo i primi $2t$ coefficienti perché

$$e(\alpha^i) = b(\alpha^i) - a(\alpha^i) = b(\alpha^i)$$

in quanto a appartiene al codice e dunque si annulla in α^i per $i \leq 2t$ e conosciamo $\omega/\sigma \pmod{x^{2t+1}}$.[2] Il polinomio locatore è il polinomio di grado minimo che soddisfa la relazione modulo x^{2t+1}. Infatti, consideriamo l'equazione

$$\omega(x) = \Big(\sum_{i=1}^{2t} e(\alpha^i) x^i\Big)\sigma(x) \pmod{x^{2t+1}}$$

Chiamato $e = |M|$, scriviamo $\sigma(x) = \sum_{i=0}^{e} \sigma_i x^i$. Allora, dato che $\deg(\omega) \leq e \leq t$, si ha

$$\sum_{i=0}^{k-1} \sigma_i e(\alpha^{k-i}) = 0 \quad \text{per } e+1 \leq k \leq 2t$$

Chiaramente σ soddisfa la relazione; sia $\tilde{\sigma} = \sum_{i=0}^{e} \tilde{\sigma}_i x^i$ una soluzione di grado minimo. Allora

$$\begin{aligned}\sum_{i=0}^{k-1} \tilde{\sigma}_i e(\alpha^{k-i}) &= \sum_{i=0}^{k-1} \tilde{\sigma}_i \sum_{l\in M} e_l \alpha^{(k-i)l} \\ &= \sum_{l\in M} e_l \alpha^{kl} \sum_{i=0}^{k-1} \tilde{\sigma}_i \alpha^{-il} \\ &= \sum_{l\in M} e_l \alpha^{kl} \tilde{\sigma}(\alpha^{-l}) = 0\end{aligned}$$

dove l'ultimo passaggio è motivato dal fatto che $k-1 \geq e$. Dunque $\tilde{\sigma}(\alpha^{-l})$ risolve il sistema lineare la cui matrice dei coefficienti è $(e_l \alpha^{kl})_{l,k}$. Ma la matrice del sistema con $e+1 \leq 2t-e+1 \leq k \leq 2t$ e $l \in M$, è invertibile poiché è una Vandermonde moltiplicata per diagonali non singolari, e $e_l \neq 0$. Di conseguenza $\tilde{\sigma}(\alpha^{-l}) = 0$ per $l \in M$. Questo significa che $\tilde{\sigma}$ appartiene all'ideale generato da σ ($\tilde{\sigma}$ si annulla negli zeri di σ) e dunque $\deg(\tilde{\sigma}) \geq \deg(\sigma)$.

[2] Notiamo che σ é invertibile sia nell'anello delle serie formali, che in modulo x^{2t+1}, poiché ha termine noto non nullo

Notiamo che abbiamo mostrato un po' di piú di questo: chiamato $s(x) = \sum_{i=1}^{2t} e(\alpha^i)x^i$ il polinomio delle sindromi, allora per ogni $a(x), b(x) \in K[x]$ con $\deg(b), \deg(a) \leq 2t - e$ si ha

$$b(x) \equiv a(x)s(x) \pmod{x^{2t+1}} \implies \sigma(x)|a(x)$$

Troviamo ora un metodo per determinare σ e ω. Consideriamo l'applicazione

$$\begin{array}{rccc} \varphi\colon & K[x] \oplus K[x] & \longrightarrow & K[x]/x^{2t+1} \\ & (a(x), b(x)) & \longmapsto & a(x)s(x) - b(x) \end{array}$$

Il nucleo di φ è generato da $(0, x^{2t+1})$ e $(1, s(x))$. Dunque dobbiamo cercare σ e ω tra il generato da questi polinomi. Applichiamo l'algoritmo di gcd euclideo a $(s(x), x^{2t+1})$ fino a quando il resto non ha grado minore o uguale a $2t - e$. A quel punto, abbiamo trovato una soluzione ammissibile

$$b(x) \equiv a(x)s(x) \pmod{x^{2t+1}} \qquad \deg(b) \leq 2t - e$$

chiamiamo $\widetilde{b}(x)$ il resto subito precedente a $b(x)$ nell'algoritmo, e dunque tale che $\deg \widetilde{b}(x) > 2t - e$. Per proprietà dell'algoritmo euclideo, avremo che

$$\deg(a) + 2t - e < \deg(a) + \deg(\widetilde{b}) = 2t + 1 \implies \deg(a) \leq e \leq 2t - e$$

e per quanto mostrato sopra, $\sigma(x)|a(x)$, ma dato che hanno lo stesso grado, $a(x)$ deve coincidere con σ a meno di scalare, e di conseguenza $b(x)$ deve coincidere con ω.

Interpolazione e FFT Sia K un campo finito e sia ω una radice primitiva n-esima dell'unità. Possiamo considerare l'applicazione

$$\begin{array}{rccc} \varphi\colon & K[x]/(x^n - 1) & \longrightarrow & K^n \\ & p(x) & \longmapsto & (p(1), p(\omega), \ldots, p(\omega^{n-1})) \end{array}$$

Possiamo interpretare K^n come $K[x]/(x^n - 1)$ mediante l'identificazione

$$\begin{array}{rccc} \psi\colon & K^n & \longrightarrow & K[x]/(x^n - 1) \\ & (a_0, \ldots, a_{n-1}) & \longmapsto & \sum_{i=0}^{n-1} a_i x^i \end{array}$$

e questo ci permette di riapplicare un'altra volta la prima mappa. Vediamo cosa succede sui monomi:

$$\begin{aligned} x^r &\overset{\varphi}{\longmapsto} (1, \omega^r, \ldots, \omega^{r(n-1)}) \\ &\overset{\psi}{\longmapsto} \sum_{i=0}^{n-1} \omega^{ri} x^i \\ &\overset{\varphi}{\longmapsto} \Big(\sum_{i=0}^{n-1} \omega^{ri}, \ldots, \sum_{i=0}^{n-1} \omega^{r(n-1)+i(n-1)} \Big) \\ &= (0, \ldots, 0, n, 0, \ldots, 0) \\ &\overset{\psi}{\longmapsto} nx^{n-r} \end{aligned}$$

Dunque, per linearità, abbiamo che $\psi\varphi\psi\varphi(f)$ coincide con n volte il reciproco di f, a meno del termine noto. In particolare, chiamato a_0 il termine noto,

$$\psi\varphi\psi\varphi(f) = nx^n(f(1/x) - a_0) + na_0$$

Questa operazione è lineare, e se $\phi = \psi\varphi$, otteniamo che ϕ^4 è la moltiplicazione per n^2, dunque se n è coprimo con la caratteristica di K, ϕ è un isomorfismo.

Preso ora un intero $0 < d < n$, consideriamo

$$C(n,d) = \varphi\left(p(x) \in {}^{K[x]}\!/_{(x^n-1)} \;\middle|\; \deg p(x) \leq n-d\right)$$

Questo è un codice, ma invece di studiarne le caratteristiche, ci accorgiamo che in realtà è un codice BCH.

Poniamo $K = \mathbb{F}_q$, e prendiamo f di grado $\leq n-d$. Questo ha i coefficienti da $n-d+1$ a $n-1$ nulli, dunque $\phi^2(f)$ ha i coefficienti da $d-1$ a 1 nulli. Ciò vuol dire che $\phi(f)$ si annulla su $\omega, \omega^2, \ldots, \omega^{d-1}$, e dunque $\varphi(f)$ è sia un elemento di $C(n,d)$, che del codice BCH su $\mathbb{F}_q$ di lunghezza n, distanza d, ed elementi $\omega, \omega^2, \ldots, \omega^{d-1}$. Se adesso n è coprimo con q, si ottiene che i due codici sono uguali.

La scrittura in polinomi di grado $\leq n-d$ è pertanto una maniera più comoda per rappresentare un codice BCH in senso stretto, e grazie alla velocità della FFT, è possibile passare da un codice all'altro in poco tempo.

1.2 Codici di Goppa

Abbiamo visto che i codici BCH in senso stretto su $\mathbb{F}_q$, hanno matrice di parità su $\mathbb{F}_{q^n}$ del tipo

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha & \alpha^2 & \cdots & \alpha^r \\ \vdots & \vdots & & \vdots \\ \alpha^{q-2} & \alpha^{2(q-2)} & \cdots & \alpha^{r(q-2)} \end{pmatrix}$$

Se moltiplichiamo la i-esima riga della matrice per un elemento $h_i \in \mathbb{F}_{q^n}$ la distanza del codice su $\mathbb{F}_{q^n}$ non varia, ma potrebbe variare quella della sua contrazione a $\mathbb{F}_q$, così come potrebbe variare la dimensione. I codici BCH sono normalmente non buoni dal punto di vista della disuguaglianza di Gilbert-Varshamov e questo non li rende ottimali. Si è allora pensato di utilizzare l'osservazione di sopra per ottenere codici migliori e con la stessa facilità di decodifica dei codici BCH. Scriviamo allora in un altro modo la condizione di appartenenza a un codice BCH. Sia β una radice n-esima primitiva dell'unità e sia C il codice BCH su $\mathbb{F}_q$ associato di lunghezza n e distanza designata d. Sia c un elemento del codice. Allora

$$\sum_{i=0}^{n-1} c_i \beta^{ik} = 0$$

per $k = 1, \dots, d-1$. Notiamo che

$$\begin{aligned}\frac{z^n - 1}{z - \beta^{-i}} &= z^{n-1} + z^{n-2}\beta^{-i} + \cdots + z\beta^{-i(n-2)} + \beta^{-i(n-1)} \\ &= \sum_{j=0}^{n-1} z^j \beta^{-i(n-j-1)} \\ &= \sum_{j=0}^{n-1} z^j \beta^{i(j+1)}\end{aligned}$$

Dunque

$$\begin{aligned}\sum_{i=0}^{n-1} \frac{c_i(z^n - 1)}{z - \beta^{-i}} &= \sum_{i=0}^{n-1} c_i \Big(\sum_{j=0}^{n-1} z^j \beta^{i(j+1)} \Big) \\ &= \sum_{j=0}^{n-1} z^j \sum_{i=0}^{n-1} c_i \beta^{i(j+1)} \\ &= \sum_{j=d-1}^{n-1} z^j \sum_{i=0}^{n-1} c_i \beta^{i(j+1)} \\ &= p(z) z^{d-1}\end{aligned}$$

Di conseguenza,

$$\sum_{i=0}^{n-1} \frac{c_i}{z - \beta^{-i}} = \frac{p(z) z^{d-1}}{z^n - 1}$$

Questa forma suggerisce allora la seguente definizione:

Definizione 1.14. Sia $g \in \mathbb{F}_{q^r}[x]$ un polinomio monico di grado $t \leq n$. Siano $L = \{\gamma_0, \dots, \gamma_{n-1}\} \subseteq \mathbb{F}_{q^r}$ un insieme di elementi distinti tali che $g(\gamma_i) \neq 0$ per ogni i. Chiamiamo $\Gamma(L, g)$ il codice di Goppa i cui elementi $c = (c_0, \dots, c_{n-1}) \in (\mathbb{F}_q)^n$ soddisfano

$$\sum_{i=0}^{n-1} \frac{c_i}{x - \gamma_i} \equiv 0 \pmod{g}$$

Intanto, notiamo che il codice è ben definito. Infatti, $x - \gamma_i$ è invertibile (mod g), in quanto $\gcd(x - \gamma_i, g) = 1$ (per ipotesi, γ_i non è una radice di g.) In particolare,

$$\frac{1}{z - \gamma} \equiv -\frac{1}{g(\gamma)} \frac{(g(z) - g(\gamma))}{z - \gamma} \pmod{g}$$

e il secondo membro è un polinomio in quanto $g(z) - g(\gamma)$ si annulla in γ e dunque appartiene all'ideale $(z - \gamma)$.

Inoltre, i codici di Goppa sono una generalizzazione dei codici BCH nel senso che, se poniamo $g = z^{d-1}$ e $L = \{1, \beta, \dots, \beta^{n-1}\}$, il codice di Goppa $\Gamma(L, g)$ coincide con il codice BCH in senso stretto generato di lunghezza n e distanza designata d su $\mathbb{F}_{q^n}$.

Cerchiamo ora una matrice di parità. Lavoriamo su $\mathbb{F}_{q^n}$ per poi trarre informazioni sul codice contratto a $\mathbb{F}_q$. Dalla definizione del codice e dall'espressione

per l'inverso di $x - \gamma_i$, otteniamo

$$\sum_{i=0}^{n-1} \frac{c_i}{x-\gamma_i} \equiv -\sum_{i=0}^{n-1} c_i \frac{1}{g(\gamma_i)} \frac{(g(z)-g(\gamma_i))}{z-\gamma_i} \pmod{g}$$

Riscriviamo in maniera più chiara la frazione che compare nella sommatoria. Chiamiamo per brevità $g(\gamma_j)^{-1} = h_j$ e scriviamo $g(z) = \sum_{i=0}^{t} g_i z^i$. Allora

$$\begin{aligned}\frac{g(z)-g(\gamma_l)}{z-\gamma_l} &= \sum_{i=0}^{t} g_i \frac{z^i - \gamma_l^i}{z-\gamma_l} \\ &= \sum_{i=0}^{t} g_i \sum_{j=0}^{i-1} z^{i-1-j}\gamma_l^j \\ &= \sum_{i+j\le t-1} g_{i+j+1} z^i \gamma_l^j\end{aligned}$$

Dunque la condizione di appartenenza al codice diventa

$$\sum_{i=0}^{n-1} c_i h_i \sum_{j+k\le t-1} g_{j+k+1} z^j \gamma_i^k \pmod{g(z)}$$

ossia

$$\sum_{i=0}^{n-1} c_i h_i \sum_{k=0}^{t-1-j} g_{j+k+1}\gamma_i^k = 0 \quad j = 0, \dots t-1$$

La matrice di parità associata a $\Gamma(L, g)$ è allora

$$A = \begin{pmatrix} h_0 g_t & h_0(g_{t-1} + \gamma_0 g_t) & \cdots & h_0 \sum_{k=0}^{t-1} g_{k+1}\gamma_0^k \\ h_1 g_t & h_1(g_{t-1} + \gamma_1 g_t) & \cdots & h_1 \sum_{k=0}^{t-1} g_{k+1}\gamma_1^k \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-1} g_t & h_{n-1}(g_{t-1} + \gamma_{n-1} g_t) & \cdots & h_{n-1} \sum_{k=0}^{t-1} g_{k+1}\gamma_{n-1}^k \end{pmatrix}$$

Tramite eliminazione gaussiana, possiamo trovare la fattorizzazione

$$A = \begin{pmatrix} h_0 & & & \\ & h_1 & & \\ & & \ddots & \\ & & & h_{n-1} \end{pmatrix} \begin{pmatrix} 1 & \gamma_0 & \cdots & \gamma_0^{t-1} \\ 1 & \gamma_1 & \cdots & \gamma_1^{t-1} \\ \vdots & \vdots & & \vdots \\ 1 & \gamma_{n-1} & \cdots & \gamma_{n-1}^{t-1} \end{pmatrix} \begin{pmatrix} g_t & g_{t-1} & \cdots & g_1 \\ 0 & g_t & \cdots & g_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & g_t \end{pmatrix}$$

e dunque ci siamo ricondotti a una matrice simile a quella dei codici BCH, in cui ogni riga è moltiplicata per un elemento di $\mathbb{F}_{q^r}$, a meno di moltiplicazione per la matrice a destra, che comunque è invertibile. Di conseguenza, ritroviamo tutti i risultati sui codici BCH:

Teorema 1.15. Sia $\Gamma(L, g)$ un codice di Goppa, dove $g \in \mathbb{F}_{q^r}[x]$. Allora la distanza di $\Gamma(L, g)$ è $\ge \deg g + 1$ e la sua dimensione è $\ge |L| - r \deg g$.

Occupiamoci ora della decodifica, che risulta praticamente uguale a quella dei codici BCH. Sia M l'insieme degli indici per il quale la componente della sindrome è non nulla. Consideriamo il polinomio errore, il polinomio locatore e il polinomio valutatore

$$s(x) = \sum_{i \in M} \frac{e_i}{x - \gamma_i} \qquad \sigma(z) = \prod_{i \in M} z - \gamma_i \qquad \omega(x) = \sum_{i \in M} e_i \prod_{\substack{j \neq i \\ j \in M}} z - \gamma_j$$

Dunque vale l'equazione chiave $\sigma(z)s(z) = \omega(z) \pmod g$, che si risolve come nel caso dei codici BCH. L'unica cosa da notare è che σ è la soluzione di grado minimo. Infatti, siano $\tilde{\sigma}$ e $\tilde{\omega}$ due polinomi tali che $\tilde{\omega}(x) = s(x)\tilde{\sigma}(x)$ e supponiamo $\deg(\tilde{\sigma}) < \deg(\sigma)$. Allora, dalle relazioni

$$\tilde{\omega}(x) = s(x)\tilde{\sigma}(x) \pmod{g(x)} \qquad \omega(x) = s(x)\sigma(x) \pmod{g(x)}$$

segue che $\sigma(x)\tilde{\omega}(x) - \tilde{\sigma}(x)\omega = 0 \pmod{g(x)}$. Tali polinomi hanno però grado $\leq d = \deg(g)$ e dunque il primo membro deve essere il polinomio nullo; dato che $(\sigma, \omega) = 1$, ne segue che $\sigma \mid \tilde{\sigma}$, da cui l'uguaglianza tra σ e $\tilde{\sigma}$.

I codici di Goppa hanno avuto una certa rilevanza in crittografia. Consideriamo infatti un codice di Goppa $\Gamma(L, g)$ binario di lunghezza alta (dell'ordine di 1000). Se il mittente e il destinatario conoscono la matrice di parità, si può appositamente introdurre degli errori nella trasmissione del messaggio, minori della metà della distanza, in modo tale che solo il destinatario sia in grado di ricostruire il messaggio.

1.3 Codici di Reed-Muller

I codici di Reed-Muller sono codici lineari su $\mathbb{F}_q$ legati alle valutazioni dei polinomi sullo spazio affine. Per semplicità supporremo in questo caso di lavorare su $\mathbb{F}_2$. Consideriamo quindi lo spazio vettoriale $(\mathbb{F}_2)^s$ ed enumeriamone gli elementi

$$(\mathbb{F}_2)^s = \{v_1, \ldots, v_{2^s}\}$$

Dato un polinomio $f \in \mathbb{F}_2[x_0, \ldots, x_{s-1}]$, possiamo associargli il vettore di $\mathbb{F}_2^{2^s}$ costituito da $(f(v_1), \ldots, f(v_{2^s}))$.

Definizione 1.16. Il codice di Reed-Muller $\mathcal{R}(m, r)$ di lunghezza 2^m e ordine r è il sottospazio di $(\mathbb{F}_2)^{2^m}$ dato dai vettori

$$(f(v_1), \ldots, f(v_{2^m}))$$

al variare di f tra i polinomi di grado $\leq r$ in $\mathbb{F}_2[x_0, \ldots, x_{m-1}]$.

Studiamo la dimensione di un codice di Reed-Muller. Indichiamo i polinomi di grado $\leq r$ con $\mathbb{F}_2[x_0, \ldots, x_{m-1}]_{\leq r}$. Consideriamo l'applicazione lineare

$$\begin{array}{rccc} \varphi\colon & \mathbb{F}_2[x_0, \ldots, x_{m-1}] & \longrightarrow & \mathbb{F}_2^{2^m} \\ & f & \longmapsto & (f(v_1), \ldots, f(v_{2^m})) \end{array}$$

Allora $\mathrm{Ker}(\varphi)$ è l'ideale generato da $(x_i^2 - x_i)$, [3] e dunque il codice ha dimensione uguale al numero di monomi liberi da quadrati di grado $\leq r$:

$$1 + \binom{m}{1} + \binom{m}{2} + \cdots + \binom{m}{r}$$

[3] per dimostrarlo, la maniera piú semplice é per induzione sul numero di variabili

Esempio. Il codice $\mathcal{R}(m,0)$ è formato dai vettori $(0,\dots,0)$ e $(1,\dots,1)$ e dunque è il codice di ripetizione. Il codice $\mathcal{R}(m,m)$ è invece il codice formato da tutti gli elementi di $\mathbb{F}_2^{2^m}$ perché

$$\sum_{i=0}^{m} \binom{m}{i} = 2^m$$

e dunque la dimensione del codice è uguale alla dimensione dello spazio vettoriale.

Cerchiamo ora di capire quale sia la distanza di $\mathcal{R}(m,r)$. Intanto, notiamo che il polinomio $f = x_0 x_1 \dots x_{r-1}$ è tale che $\varphi(f)$ abbia $2^m - 2^{m-r}$ componenti nulle. Questo mostra che la distanza di un codice di Reed-Muller $\mathcal{R}(m,r)$ è $\leq 2^{m-r}$. Per mostrare l'altra disuguaglianza, utilizziamo il seguente lemma:

Lemma 1.17. Sia $f \in \mathbb{F}_2[x_0,\dots,x_{m-1}]$ di grado s (se $f=0$, si può considerare $s = +\infty$). Allora $\#V(f) \leq 2^m - 2^{m-s}$.

Osserviamo immediatamente che, se $\tilde{s} \leq s$, allora $2^m - 2^{m-\tilde{s}} \leq 2^m - 2^{m-s}$, quindi il risultato si applica anche a f di grado $\leq s$.

Dimostrazione. Dimostriamo prima il risultato per gli f con monomi squarefree, per induzione. Per $m=1$ il risultato è ovvio ($f \in \{0,\ 1,\ x,\ x+1\}$). Supponendo la tesi vera per m, osserviamo che, per f con monomi squarefree, si può sempre scrivere

$$f(x_0,\dots,x_m) = x_m f_1(x_0,\dots,x_{m-1}) + (1-x_m) f_0(x_0,\dots,x_{m-1}),$$

dove $f_\ell(x_0,\dots,x_{m-1}) = f(x_0,\dots,x_{m-1},\ell)$. Si osserva che $\#V(f) = \#V(f_0) + \#V(f_1)$, e banalmente $\#V(f_0), \#V(f_1) \leq 2^{m-1}$. Se $f_0, f_1 \neq 0$, allora per ipotesi induttiva $\#V(f_0), \#V(f_1) \leq 2^{m-1} - 2^{m-s-1}$, da cui $\#V(f) \leq 2^m - 2^{m-s}$. Se wlog $f_1 = 0$, in particolare $\deg(f_0) \leq s-1$ e per ipotesi induttiva $\#V(f_0) \leq 2^{m-1} - 2^{m-s}$ (è un polinomio di grado al più $s-1$ in $m-1$ variabili), da cui $\#V(f) \leq 2^{m-1} - 2^{m-s} + 2^{m-1} = 2^m - 2^{m-s}$.

D'altra parte, se f non fosse a monomi squarefree, chiamando $\tilde{f}$ il polinomio di grado $\tilde{s} < s$ ottenuto rendendo squarefree i monomi di f, le valutazioni di f e $\tilde{f}$ coincidono, quindi $\#V(f) = \#V(\tilde{f})$. Applicando il risultato a $\tilde{f}$, otteniamo

$$\#V(f) \leq 2^m - 2^{m-\tilde{s}} < 2^m - 2^{m-s}. \qquad \square$$

Interpolazione Sfruttiamo l'interpolazione polinomiale in più variabili. Sia c il messaggio ricevuto e supponiamo di voler ricostruire il messaggio del mittente.

Per prima cosa, troviamo un polinomio con monomi liberi da quadrati tale che c sia il vettore delle valutazioni sui punti di $(\mathbb{F}_2)^m$. Questo é possibile farlo tramite interpolazione multivariata: dato $v \in (\mathbb{F}_2)^m$ chiamiamo L_v il polinomio

$$L_v(x) = (1 + x_0 + v_0)(1 + x_1 + v_1)\dots(1 + x_{m-1} + v_{m-1})$$

che ha la proprietá $L_v(x) = 1 \iff x = v$. Da questo, possiamo scrivere il polinomio interpolatore di c come

$$p_c(x) = \sum_{c_v = 1} L_v(x)$$

Notiamo che $p_c(x)$ é un polinomio con monomi liberi da quadrati, dunque é l'unico di questo tipo che interpoli c. Se il suo grado é minore o uguale ad r, allora abbiamo ricostruito il messaggio originale.

Notiamo anche che imporre $p_c(v) = c_v$ é una condizione lineare sui coefficienti di $p_c(x)$, dunque per ricostruire un polinomio del codice, basta imporre tante condizioni quanto é la dimensione del codice.

Definizione Ricorsiva I codici di Reed Muller si possono caratterizzare anche in maniera ricorsiva. Abbiamo giá detto che $\mathcal{R}(m,0)$ sono i codici di ripetizione, e $\mathcal{R}(m,k) \cong \mathbb{F}_2^m$ per $k \geq m$. Cerchiamo di ricavare $\mathcal{R}(m,k)$ con $m > k$ in relazione a quelli con k minori.

Preso un polinomio $p(x)$ con monomi squarefree su $\mathbb{F}_2[x_0, \ldots, x_{m-1}]$ di grado minore o uguale a k, lo spezziamo nei monomi che contengono x_0, e quelli che non lo contengono, ossia $p(x) = p_0(x) + x_0 p_1(x)$. Il relativo messaggio del codice $\mathcal{R}(m,k)$ puó essere spezzato in due parti: le valutazioni con $x_0 = 0$ e quelle con $x_1 = 1$, indicandole con $c = (c_0, c_1)$.

- $p_0(x)$ non contiene x_0, dunque le sue valutazioni non dipendono dalla prima variabile, dunque saranno del tipo (α, α), ma visto che $p_0(x)$ puó essere visto come un polinomio in $\mathbb{F}_2[x_1, \ldots, x_{m-1}]$ di grado minore o uguale a k, allora α appartiene a $\mathcal{R}(m-1,k)$.
- le valutazioni di $x_0 p_1(x)$ saranno del tipo $(0, \beta)$, ma visto che $p_1(x)$ non contiene x_0, puó essere visto come un polinomio in $\mathbb{F}_2[x_1, \ldots, x_{m-1}]$ di grado minore o uguale a $k-1$, dunque β appartiene a $\mathcal{R}(m-1,k-1)$.

Questo ci dice che le valutazioni di $p(x)$ sono $(\alpha, \alpha+\beta)$, con $\alpha \in \mathcal{R}(m-1,k)$ e $\beta \in \mathcal{R}(m-1,k-1)$. Questo ci dá una definizione equivalente:

Definizione 1.18. I codici di Reed Muller $\mathcal{R}(m,k)$ sono definiti come

- $\mathcal{R}(m,0)$ sono i codici di ripetizione di lunghezza 2^m.
- se $k \geq m$, allora $\mathcal{R}(m,k)$ é isomorfo a $\mathbb{F}_2^k$.
- $\mathcal{R}(m,k)$ con $k < m$ é ricavato ricorsivamente come

$$\{v \in \mathbb{F}_2^m \,:\, v = (\alpha, \alpha+\beta),\ \alpha \in \mathcal{R}(m-1,k),\ \beta \in \mathcal{R}(m-1,k-1)\}$$

Notiamo che $\mathcal{R}(m-1,k-1) \subseteq \mathcal{R}(m-1,k)$, dunque ogni elemento del codice $\mathcal{R}(m,k)$ é composto da due stringhe di $\mathcal{R}(m-1,k)$, da cui la distanza di $\mathcal{R}(m,k)$ é almeno il doppio di quella di $\mathcal{R}(m-1,k)$, e dato che ci sono tutte le stringhe del tipo (α,α), allora la distanza é esattamente il doppio. Dato che $\mathcal{R}(k,k) \cong \mathbb{F}_2^k$ ha distanza 1, allora

$$\begin{aligned} dist(\mathcal{R}(m,k)) &= 2 \cdot dist(\mathcal{R}(m-1,k)) = 2^2 \cdot dist(\mathcal{R}(m-2,k)) = \ldots \\ &= 2^{m-k} \cdot dist(\mathcal{R}(m-(m-k),k)) = 2^{m-k} \cdot dist(\mathcal{R}(k,k)) = 2^{m-k} \end{aligned}$$

Come abbiamo detto all'inizio, possiamo generalizzare questi codici ad $\mathbb{F}_q$ con $q = p^n$. In questo caso, il codice $\mathcal{R}(m,k)$ diviene di lunghezza q^m, e se $k \geq (q-1)m$, allora $\mathcal{R}(m,k) \cong \mathbb{F}_q^m$. Analogamente a sopra, si puó scrivere una definizione ricorsiva, e si ottiene la distanza per $k < (q-1)m$

$$dist(\mathcal{R}(m,k)) = \begin{cases} q^{m-h} & \text{se } k = (q-1)h \\ q^{m-h}(q-1) & \text{se } h-1 < \frac{k}{q-1} < h \end{cases}$$

Capitolo 2

Algoritmi

In questa sezione, studiamo alcuni algoritmi base che sono utili per l'implementazione di codici e per comprendere la difficoltà di alcuni problemi rilevanti in crittografia.

2.1 Fattorizzazione di polinomi

Occupiamoci ora della fattorizzazione di polinomi a coefficienti in $\mathbb{F}_p$. Sia $f \in \mathbb{F}_p[x]$ un polinomio di grado n; possiamo chiaramente supporlo monico (a meno di invertire il coefficiente di testa) e squarefree. Per fattorizzazione unica, f si scrive in modo unico come prodotto di fattori irriducibili

$$f = \prod_{i=1}^{k} f_i$$

dove $(f_i, f_j) = 1$. Per il teorema cinese del resto, vale allora

$$\mathbb{F}_p[x]/(f) \sim \prod_{i=1}^{k} \mathbb{F}_p[x]/(f_i)$$

e dunque il quoziente è un prodotto di campi finiti. Dato $g \in \mathbb{F}_p[x]/(f)$, possiamo allora vederlo come un elemento del prodotto

$$g = (g_1, \ldots, g_k) \in \prod_{i=1}^{k} \mathbb{F}_p[x]/(f_i)$$

Supponiamo che $g_1 = 0$ e che $g_i \neq 0$ per ogni indice $i \geq 2$. Allora $(g, f) = f_1$; possiamo così dividere f per f_1 e iterare. L'idea dell'algoritmo è quella di trovare degli opportuni g_i in modo tale che $(g_i, f) = f_i$. Il problema rimane chiaramente come trovare tali g_i. L'endomorfismo di Frobenius è una ben definita applicazione lineare tra $\mathbb{F}_p$-spazi vettoriali:

$$\varphi_p \colon \quad \mathbb{F}_p[x]/(f) \quad \longrightarrow \quad \mathbb{F}_p[x]/(f)$$

Notiamo che se $a \in \mathbb{F}_p$, allora $\varphi_p(a) = a^p = a$, ma per calcolare esattamente il fissato di φ_p, dobbiamo notare che

$$g = (g_1, \dots, g_k) \in \prod_{i=1}^{k} \mathbb{F}_p[x]/(f_i)$$

$$\varphi_p(g) = g \iff (g_1^p, \dots, g_k^p) = (g_1, \dots, g_k) \iff g_i^p = g_i \ \forall i$$

Ma in $\mathbb{F}_{p^n} \simeq \mathbb{F}_p[x]/(f_i)$, $\mathrm{Fix}(\varphi_p) = \mathbb{F}_p$, poiché gli unici elementi che soddisfano $x^p - x$ sono esattamente $\mathbb{F}_p$. Dunque $\mathrm{Fix}(\varphi_p) = \mathbb{F}_p^k$, poiché ne prendiamo una copia da ogni componente. Di conseguenza, abbiamo trovato un modo per trovare il numero di fattori irriducibili di f: è sufficiente calcolare la dimensione del nucleo dell'omomorfismo $\varphi_p - Id$. Per questo, è sufficiente scrivere la matrice che rappresenta l'omomorfismo nella base $1, x, \dots x^{n-1}$ e applicare l'algoritmo di Gauss.

Osservazione 2.1. Notiamo che se volessimo fattorizzare f su $\mathbb{F}_{p^n}$, l'unica modifica da effettuare sarebbe quella di considerare il Frobenius $\varphi \colon x \longmapsto x^{p^n}$.

Bisogna ora escogitare un espediente per calcolare effettivamente i fattori irriducibili. Sia $g \in \mathrm{Fix}(\varphi_p)$. Possiamo vedere allora g come elemento del prodotto $g \leftrightarrow (s_1, \dots, s_k)$ dove ogni s_i appartiene a $\mathbb{F}_p$. Fissato un indice i, supponiamo che $s_i \neq s_j$ quando $i \neq j$. Allora $g - s_i$ ha solo la i-esima componente uguale a 0 e dunque vale

$$f_i \mid g - s_i \qquad\qquad f_j \nmid g - s_i \ \ \forall j \neq i$$

Quindi, $(g - s_i, f) = (g - s_i, f_i) = f_i$. Abbiamo così trovato un modo per determinare le componenti irriducibili. Ovviamente non sappiamo se esistano degli s_i unici, ma in ogni caso, questo procedimento permette di fattorizzare il polinomio, e dato che sappiamo il suo numero di componenti irriducibili, sappiamo quando fermarci.

Notiamo che l'unico caso in cui questa procedura fallisce è se g ha tutte le componenti s_i uguali, ma questo vorrebbe dire che $g \in \mathbb{F}_p$, dunque si ovvia a ciò prendendo g fuori da $\mathbb{F}_p$.

Prima di illustrare lo pseudocodice, osserviamo che detta M_φ la matrice che che rappresenta l'omomorfismo di Frobenius φ_p, necessariamente la prima colonna di $M_\varphi - Id$ è sempre nulla e dunque $rk(M_\varphi - \mathrm{Id}) < n$. Dunque, se il rango è $n - 1$, il polinomio è irriducibile, ed è inutile continuare con il resto dell'algoritmo.

Supponiamo quindi di avere in input un polinomio $f \in \mathbb{F}_p[x]$ monico e squarefree di grado n. In output, forniremo la lista dei fattori irriducibili di f.

Algoritmo 2.1 Fattorizzazione di polinomi su campi finiti

1: Inizializzare la lista dei fattori irriducibili $Irr = [\,]$
2: Costruire la matrice M_φ
3: Calcolare $r = rk(M_\varphi - \mathrm{Id})$
4: **if** $r = n - 1$ **then return** $[f]$
5: **else**
6: Calcolare una base di $\mathrm{Fix}(\varphi_p)$ $\{v_1 = 1, \ldots, v_k\}$
7: Scegliere $v \in \mathrm{Fix}(\varphi_p) \setminus \mathbb{F}_p$
8: **for** $s = 0, \ldots, p-1$ **do**
9: $g = (f, v - s)$
10: **if** $(g \neq 1) \wedge (f \neq g)$ **then**
11: $Irr = [g, Irr]$
12: $f = \frac{f}{g}$
13: **end if**
14: **end for**
15: **end if**
16: **if** $\#Irr = k$ **then**
17: **return** Irr
18: **else**
19: Ripeti l'algoritmo su ogni fattore
20: **end if**

Supponiamo che p sia un primo dispari, e consideriamo un elemento g nel fissato di φ_p. Avremo che $g^p - g \equiv 0 \pmod f$, ma allora

$$f \mid g^p - g = g(g^{\frac{p-1}{2}} - 1)(g^{\frac{p-1}{2}} + 1)$$

ossia ogni componente irriducibile di f divide esattamente uno dei tre fattori, e sappiamo anche dire quale: se $g = (g_1, \ldots, g_k)$, allora

$$f_i \text{ divide } \begin{cases} g & \text{se } g_i = 0 \\ g^{\frac{p-1}{2}} - 1 & \text{se } g_i^{\frac{p-1}{2}} - 1 = 0 \\ g^{\frac{p-1}{2}} + 1 & \text{se } g_i^{\frac{p-1}{2}} + 1 = 0 \end{cases}$$

In particolare, questo divide anche le classi di resto modulo p tra i residui quadratici e i non residui quadratici, ossia esattamente a metà se non consideriamo lo 0. Ciò vuol dire che possiamo provare a calcolare il gcd tra f e i tre fattori sopra per vedere se si scompone.

Se infatti poniamo f non irriducibile, e proviamo a fare $h = (f, g^{\frac{p-1}{2}} - 1)$, la probabilità che vada male è quando $h = f$ o 1. Ma questo succede solo quando tutti i g_i sono residui quadratici, o quando non lo sono, dunque la probabilità che $h \neq f$ o 1 è

$$P = 1 - \left(\frac{p-1}{2p}\right)^k - \left(\frac{p+1}{2p}\right)^k \geq \frac{4}{9}$$

Se proviamo a fare il gcd con tutti e tre i fattori, ci va male solo se tutti i g_i sono residui quadratici, non residui quadratici, o tutti zeri. La probabilità che ci vada bene sale a

$$P = 1 - 2\left(\frac{p-1}{2p}\right)^k - \left(\frac{1}{p}\right)^k \geq \frac{2}{3}$$

e provando con più g è molto probabile riuscire a fattorizzare f.

Per quanto riguarda il costo computazionale, il calcolo della matrice M_φ costa $O(pn^2)$ operazioni. Infatti, se $f = x^n + \sum a_i x^i$, si ha la relazione

$$x^n = -a_{n-1}x^{n-1} + \cdots + a_0$$

Supponiamo di aver calcolato $x^s = b_{n-1}x^{n-1} + \cdots + b_0$ modulo (f). Allora

$$x^{s+1} = b_{n-1}(-a_{n-1}x^{n-1} + \cdots + a_0) + b_{n-2}x^{n-1} + \cdots + b_0$$

e poichè dobbiamo ripetere questa operazione per np volte, il costo totale è $O(pn^2)$. Il calcolo del rango, con Gauss, è dell'ordine di $O(n^3)$: il calcolo di tutti i massimi comuni divisori è dell'ordine di $O(pkn^2)$. Di conseguenza, la complessità dell'algoritmo è $O(n^3 + pkn^2)$. Poiché di solito il primo scelto è molto maggiore di n, il secondo termine domina e dunque la complessità è maggiore di $O(n^3)$.

Fattorizzazione in Gradi Distinti Mostriamo ora un altro algoritmo per la fattorizzazione in $\mathbb{F}_p[x]$, basato sui seguenti due lemmi di carattere teorico:

Proposizione 2.2. Sia $q(x) \in \mathbb{F}_p[x]$ un polinomio irriducibile di grado d. Allora $q(x) \mid x^{p^d} - x$.

Dimostrazione. Sappiamo che $\mathbb{F}_p[x]/(q) \simeq \mathbb{F}_{p^d}$. Dato che per ogni elemento $a \in \mathbb{F}_{p^d}$ vale $a^{p^d} = a$, e che in particolare le radici di q sono contenute in questo campo, $x^{p^d} - x = 0 \pmod q$, cioè $q \mid x^{p^d} - x$. □

Proposizione 2.3. Sia $n \in \mathbb{N}$ e siano $\{f_i \mid i \in I\}$ tutti i polinomi irriducibili in $\mathbb{F}_p[x]$ di grado un divisore di n. Allora

$$x^{p^n} - x = \prod_{i \in I} f_i$$

Dimostrazione. Mostriamo prima che se q è un polinomio irriducibile di grado d e $d \mid n$, allora $q \mid x^{p^n} - x$. Sappiamo che $\mathbb{F}_p[x]/(q)$ è un campo con p^d elementi. Ogni radice di q soddisfa quindi $\alpha^{p^d} = \alpha$. Poichè $d \mid n$, abbiamo $n = dk$. Dunque $\alpha^{p^n} = \alpha^{p^{dk}} = \alpha$. Dunque α è radice di $x^{p^n} - x$ e dunque $q \mid x^{p^n} - x$.

Supponiamo ora che q sia un polinomio irriducibile di grado d e che $q \mid x^{p^n} - x$ e mostriamo che $d \mid n$. Sia α un elemento di $\mathbb{F}_{p^n}$ che è anche una radice di q. È ben definito l'omomorfismo

$$\begin{array}{rccc} \varphi\colon & \mathbb{F}_p[x] & \longrightarrow & \mathbb{F}_{p^n} \\ & f & \longmapsto & f(\alpha) \end{array}$$

Chiaramente, q è contenuto in $\mathrm{Ker}(\varphi)$; per massimalità dell'ideale (q), abbiamo una mappa iniettiva

$$\begin{array}{rccc} \tilde{\varphi}\colon & \mathbb{F}_p[x]/(q) & \longrightarrow & \mathbb{F}_{p^n} \\ & \bar{f} & \longmapsto & f(\alpha) \end{array}$$

e dunque in $\mathbb{F}_{p^d}$ si immerge $\mathbb{F}_{p^n}$. Per il teorema sul grado delle sottoestensioni, necessariamente $d \mid n$. □

Illustriamo ora lo pseudocodice di un algoritmo di fattorizzazione non in fattori irriducibili, ma in polinomi le cui componenti irriducibili abbiano lo stesso grado; supponiamo di avere in input un polinomio $f(x) \in \mathbb{F}_p[x]$ monico e squarefree di grado n.

Algoritmo 2.2 Fattorizzazione in gradi distinti

1: $v(x) = f(x)$
2: $w(x) = x$
3: $d = 0$
4: $g_i = 1 \ \forall i$
5: **while** $(d \leq \frac{n}{2}) \wedge (v \neq 1)$ **do**
6: $\quad d = d + 1$
7: $\quad w(x) = w(x)^p \ mod \ v$
8: $\quad g_d = (w(x) - x, v(x))$
9: $\quad$ **if** $g_d \neq 1$ **then**
10: $\quad\quad v = \frac{v}{g_d}$
11: $\quad\quad w(x) = (w(x) \ mod \ v)$
12: $\quad$ **end if**
13: **end while**
14: **return** $g_1, \ldots, g_{\frac{n}{2}}$

Come anticipato, il risultato di questo algoritmo non sono polinomi irriducibili, dunque va preso solo come un algoritmo di pre-processing del polinomio. Per raggiungere il risultato finale, utilizziamo un metodo simile a quello visto sopra. Sappiamo che g_k dividono $x^{p^k} - x$, ma se poniamo p dispari, allora questo si scompone in

$$x^{p^k} - x = x(x^{\frac{p^k-1}{2}} - 1)(x^{\frac{p^k-1}{2}} + 1)$$

dunque possiamo fare il gcd con uno dei fattori, e sperare che si scomponga. Se non funziona, possiamo sostituire $x \mapsto x + s$ con $s \in \mathbb{F}_p$, che rimescola gli elementi di $\mathbb{F}_p^k$ in maniera non banale, e riprovare. Questo metodo non assicura la scomposizione, ma le probabilità di successo sono molto alte.

Elevamento a potenza Abbiamo bisogno di calcolare durante l'algoritmo l'elevamento a potenza g^n con $n \in \mathbb{N}$. Per questo, c'è un metodo computazionalmente furbo. Infatti, scritto $n = \sum e_i 2^i$ con $e_i = 0, 1$ (ossia in base 2), allora

$$g^n = \prod_{e_i=1} g^{2^i}$$

Dunque, per ottenere quanto voluto, si può eseguire il seguente:

Algoritmo 2.3 Elevamento a potenza

```
1: y = 1
2: N = n
3: z = g
4: while N ≠ 0 do
5:     if N ≡ 1 (mod 2) then
6:         y = zy
7:     end if
8:     N = ⌊N/2⌋
9:     if N ≠ 0 then
10:        z = z · z
11:    end if
12: end while
13: return y
```

Fattorizzazione di polinomi a coefficienti interi Studiamo ora un algoritmo per la fattorizzazione di polinomi a coefficienti interi. L'idea è come sempre quella di ricondursi alla fattorizzazione su campi finiti e utilizzeremo il seguente lemma:

Lemma 2.4 (di Hensel)**.** Sia $f \in \mathbb{Z}[x]$ un polinomio monico e sia $m \in \mathbb{Z}$. Supponiamo $f = g_1 h_1$ modulo m, con $(g_1, h_1) = 1$. Allora esistono unici g_2, h_2 tali che $(g_2, h_2) = 1$, $f = g_2 h_2$ modulo m^2 e $g_2 = g_1$, $h_2 = h_1$ modulo m.

Dimostrazione. Dimostreremo solo l'esistenza. Chiaramente, se tali g_2, h_2 esistono, dovrà valere

$$g_2 \equiv g_1 + mb \ (m^2) \qquad\qquad h_2 \equiv h_1 + mc \ (m^2)$$

Ci basta allora mostrare che esistono c, b che realizzano l'uguaglianza. Chiaramente, per ipotesi, varrà $f \equiv f_1 g_1 + mk \ (m^2)$ Poichè vorremmo che $f = g_2 h_2 \ (m^2)$, otteniamo

$$\begin{aligned} f \equiv & g_2 h_2 \ (m^2) \\ \equiv & (g_1 + mb)(h_1 + mc) \ (m^2) \\ \equiv & g_1 h_1 + g_1 mc + h_1 mb \ (m^2) \end{aligned}$$

Dunque

$$m(bh_1 + cg_1) \equiv mk \ (m^2) \Rightarrow bh_1 + cg_1 \equiv k \ (m)$$

Poiché $(h_1, g_1) = 1$, esistono b e c come richiesti per l'identità di Bezout.

Mostriamo ora che h_2, g_2 sono coprimi modulo m^2. Mostriamo cioè che esistono r_2, s_2 tali che $rg_2 + sh_2 = 1$. Per ipotesi, esistono r_1, s_1 tali che $r_1 g_1 + s_1 h_1 = 1 \ (m)$, dunque esiste z tale che $r_1 g_1 + s_1 h_1 = 1 + mz \ (m^2)$. Poniamo $r_2 = r_1 + mw$ e $s_2 = s_1 + my$; mostriamo che possiamo determinare w, y in modo da ottenere quanto voluto.

$$r_2 g_2 + s_2 h_2 \equiv 1 (m^2) \iff (r_1 + mw)(g_1 + mb) + (s_1 + my)(h_1 + mc) \equiv 1 \ (m^2)$$

Sviluppando i conti, otteniamo

$$wg_1 + yh_1 + r_1 b + s_1 c + z \equiv 0 \ (m)$$

e dunque $wg_1 + yh_1 \equiv -z - r_1 b - s_1 c$. Poiché s_1, r_1, z, b, c sono già fissati e $(g_1, h_1) = 1$, otteniamo la tesi. □

Chiaramente, la fattorizzazione può essere ridotta alla fattorizzazione del contenuto e di un polinomio primitivo. In più, possiamo supporre che il polinomio sia monico. Supponiamo infatti di avere un polinomio $f = a_n x^n + \ldots + a_0$ e consideriamo il cambio di variabile $x \mapsto \frac{x}{a_n}$. Otteniamo allora il polinomio

$$f\Big(\frac{x}{a_n}\Big) = a_n\Big(\frac{x}{a_n}\Big)^n + \sum_{i=0}^{n-1} a_i \frac{x^i}{a_n^i}$$

Moltiplicando per a_n^{n-1}, si ottiene il polinomio

$$g(x) = x^n + \sum_{i=0}^{n-1} a_i a_n^{n-1-i} x^i$$

e dalla fattorizzazione di g si può riottenere facilmente la fattorizzazione di f (basta ricambiare variabile).

Supponiamo di avere in input un polinomio f monico e squarefree. In output forniremo la fattorizzazione di f. C'è da dire però che questo algoritmo non termina sempre: per esempio il polinomio $x^4 + 1$ è irriducibile su $\mathbb{Z}$, ma è riducibile in ogni $\mathbb{F}_p$.

Scegliere p primo tale che $p \nmid lc(f)$
Calcolare il bound B sui coefficienti dei fattori irriducibili
Controllare che $p \nmid Ris(f, f')$; altimenti scegliere un altro primo
Fattorizzare il polinomio modulo p
if f è irriducibile in $\mathbb{F}_p[x]$ **then**
    f è irriducibile
else
    $f = \bar{f}_1 \ldots \bar{f}_k \ mod\ p$
    Sollevare $\bar{f}_1 \ldots \bar{f}_k$ modulo $p^{2r} > 2B$
    Verificare che $\bar{f}_1 \ldots \bar{f}_k$ in notazione bilanciata dividono f in $\mathbb{Z}[x]$.
Altrimenti accoppiarli e ricombinarli.
end if

2.2 Reciprocità quadratica e calcolo della radice quadrata

Occupiamoci ora del calcolo della radice quadrata di un elemento di $\mathbb{F}_p$, con p dispari. Supponiamo dapprima che $p \equiv -1 \pmod 4$; sappiamo che in questo caso -1 non è un quadrato. Allora, posto $b = a^{p+1/4}$, abbiamo $b^4 = a^{p+1} = a^2$ e dunque $b^2 = \pm a$. Se $b^2 = -a$, allora a non ha radici quadrate, in quanto in $\mathbb{Z}/p\mathbb{Z}$ -1 non è un quadrato. Se invece $p \equiv 1 \pmod 4$, non possiamo utilizzare l'algoritmo appena trattato. Consideriamo $a \in \mathbb{F}_p$ e sia $p(x) = x^2 + bx + a$ un polinomio irriducibile (discuteremo più in là la scelta di b). Allora nel campo

$$\mathbb{F}_p[x] \big/ (x^2 + bx + a)$$

il polinomio si spezza completamente e ha come radici $\bar{x}$ e $\bar{x}^p$, in quanto il Frobenius è il generatore del gruppo di Galois, oppure, più semplicemente, perché se $\bar{x}$ è radice del polinomio, allora

$$0 = \bar{x}^2 + b\bar{x} + a = (\bar{x}^2 + b\bar{x} + a)^p = \bar{x}^{2p} + b\bar{x}^p + a$$

Dunque $a = \bar{x} \cdot \bar{x}^p = \bar{x}^{p+1}$. Dunque nel quoziente la radice quadrata di a esiste e coincide con $\bar{x}^{\frac{p+1}{2}}$. Dato che la radice quadrata è unica a meno del segno, riducendo questo modulo $x^2 + bx + a$ otteniamo come resto un polinomio di grado al più 1. Se $\bar{x}^{\frac{p+1}{2}} \equiv s \pmod{x^2 + bx + a}$ con $s \in \mathbb{F}_p$, allora $s^2 = a$ e abbiamo trovato la radice quadrata. Altrimenti, non esiste una radice quadrata in $\mathbb{F}_p$. Questo algoritmo permette anche di trovare una radice quadrata in $\mathbb{Z}/n\mathbb{Z}$; è sufficiente infatti fattorizzare n e ricostruire la radice tramite teorema cinese del resto.

Occupiamoci ora di capire come trovare un b adeguato. Dato che la caratteristica è diversa da 2, per verificare che $x^2 + bx + a$ sia irriducibile possiamo utilizzare la formula risolutiva delle equazioni di secondo grado. Per verificare l'irriducibilità basta allora vedere se $b^2 - 4a$ è un quadrato in $\mathbb{F}_p$. Per questo, una prima idea è vedere se $(b^2 - 4a)^{\frac{p-1}{2}} \equiv 1$. In realtà, il simbolo di Legendre rende questa procecura più semplice:

Definizione 2.5. Sia $n \in \mathbb{N}$ e sia p un primo. Definiamo il simbolo di Legendre come

$$\left(\frac{n}{p}\right) = \begin{cases} 0 & \text{se } n \equiv 0 \pmod{p} \\ 1 & \text{se } n \text{ è un quadrato} \pmod{p} \\ -1 & \text{se } n \text{ non è un quadrato} \pmod{p} \end{cases}$$

Il simbolo di Legendre è moltiplicativo, ossia $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$. La definizione di per sé non è molto utile, ma vale la seguente:

Teorema 2.6 (Reciprocità quadratica)**.** Siano p, q primi dispari. Allora

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

Inoltre,

$$\left(\frac{2}{p}\right) = \begin{cases} 1 & \text{se } p \equiv \pm 1 \pmod{8} \\ -1 & \text{se } p \equiv \pm 3 \pmod{8} \end{cases}$$

Dimostrazione. Mostriamo prima la reciprocità sui primi diversi da 2. Consideriamo un campo $\mathbb{F}_p^s$ che contenga una radice q-esima primitiva dell'unità ζ e definiamo

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

Notiamo che $G^2 = (-1)^{\frac{q-1}{2}} q$. Infatti

$$
\begin{aligned}
G^2 &= \sum_{j=0}^{q-1} \left(\frac{j}{q}\right) \zeta^j \sum_{k=0}^{q-1} \left(\frac{-k}{q}\right) \zeta^{-k} \\
&= \Big(\sum_{j=1}^{q-1} \left(\frac{j}{q}\right) \zeta^j\Big)\Big(\sum_{k=1}^{q-1} \left(\frac{-k}{q}\right) \zeta^{-k}\Big) && \text{se } j{=}0 \text{ o } k{=}0 \text{ è } 0 \\
&= \sum_{j=1}^{q-1}\sum_{k=1}^{q-1} \left(\frac{-kj}{q}\right) \zeta^{j-k} && \text{usiamo la moltiplicatività del simbolo di Legendre} \\
&= \left(\frac{-1}{q}\right) \sum_{j=1}^{q-1}\sum_{k=1}^{q-1} \left(\frac{jk}{q}\right) \zeta^{j-k} && \text{si porta fuori dalla sommatoria } \left(\tfrac{-1}{q}\right) \\
&= \left(\frac{-1}{q}\right) \sum_{j=1}^{q-1}\sum_{k=1}^{q-1} \left(\frac{jk^2}{q}\right) \zeta^{jk-k} && \text{Dato che sono coprimi con } q \text{ sostituiamo } j \mapsto jk \\
&= \left(\frac{-1}{q}\right) \sum_{j=1}^{q-1}\sum_{k=1}^{q-1} \left(\frac{j}{q}\right) \zeta^{k(j-1)} && \left(\frac{jk^2}{q}\right) = \left(\frac{j}{q}\right)\left(\frac{k}{q}\right)^2 = \left(\frac{j}{q}\right) \\
&= \left(\frac{-1}{q}\right) \sum_{j=0}^{q-1}\sum_{k=0}^{q-1} \left(\frac{j}{q}\right) \zeta^{k(j-1)} && \text{si riaggiungono alla sommatoria } j{=}0,\ k{=}0 \\
&= \left(\frac{-1}{q}\right) \sum_{j=0}^{q-1} \left(\frac{j}{q}\right) \sum_{k=0}^{q-1} \zeta^{k(j-1)} \\
&= \left(\frac{-1}{q}\right) \left(\frac{1}{q}\right) q = \left(\frac{-1}{q}\right) q && \text{l'unico addendo non nullo è quello per cui } j{=}1
\end{aligned}
$$

Calcoliamo ora G^p in due modi diversi. Utilizzando la formula appena dimostrata, si ottiene:

$$
\begin{aligned}
G^p &= (G^2)^{\frac{p-1}{2}} G \\
&= ((-1)^{\frac{q-1}{2}} q)^{\frac{p-1}{2}} G \\
&= (-1)^{\frac{p-1}{2}\frac{q-1}{2}} G q^{\frac{p-1}{2}}
\end{aligned}
$$

D'altra parte,

$$\begin{aligned} G^p &= \sum_{j=0}^{q-1} \left(\frac{j}{q}\right) \zeta^{pj} \\ &= \sum_{j=0}^{q-1} \left(\frac{p^2 j}{q}\right) \zeta^{pj} \\ &= \left(\frac{p}{q}\right) \sum_{j=0}^{q-1} \left(\frac{pj}{q}\right) \zeta^{pj} \\ &= \left(\frac{p}{q}\right) \sum_{j=0}^{q-1} \left(\frac{j}{q}\right) \zeta^{j} \\ &= \left(\frac{p}{q}\right) G \end{aligned}$$

Di conseguenza,

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

Dato che $q^{\frac{p-1}{2}} = \left(\frac{q}{p}\right)$, si ha la tesi. Consideriamo ora il caso $\left(\frac{2}{p}\right)$ e definiamo la funzione

$$f(n) = \begin{cases} (-1)^{\frac{n^2-1}{8}} & n \equiv 1 \pmod 2 \\ 0 & n \equiv 0 \pmod 2 \end{cases}$$

f è a valori in $\{0, \pm 1\}$ in quanto uno tra $n+1$ e $n-1$ è sempre divisibile per 4. Se m, n sono dispari, allora

$$\frac{(mn)^2-1}{8} \equiv \frac{(mn)^2-1}{8} - \frac{(m^2-1)(n^2-1)}{8} = \frac{m^2-1}{8} + \frac{n^2-1}{8} \pmod 2$$

da cui si deduce che f è moltiplicativa. Sia ζ un elemento di ordine 8 di $\mathbb{F}_{p^2}$ e consideriamo l'elemento

$$G = \sum_{j=0}^{7} f(j)\zeta^j$$

Notiamo che $G \in \mathbb{F}_{p^2}$ e

$$G = \zeta - \zeta^3 - \zeta^5 + \zeta^7 = \zeta - \zeta^3 + \zeta - \zeta^3 = 2(\zeta - \zeta^3)$$

In particolare $G^2 = 8$ perché

$$G^2 = 4(\zeta - \zeta^3)^2 = 4(\zeta^2 + \zeta^6 - 2\zeta^4) = 8$$

Calcoliamo ora G^p in due modi diversi.

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

D'altra parte,

$$\begin{aligned} G^p &= \sum_{j=0}^{7} f(j)^p \zeta^{pj} \\ &= \sum_{j=0}^{7} f(j)\zeta^{pj} \\ &= \sum_{j=0}^{7} f(p^2 j)\zeta^{pj} \\ &= f(p)\sum_{j=0}^{7} f(j)\zeta^{j} \\ &= f(p)G \end{aligned}$$

Di conseguenza $f(p) = \left(\frac{2}{p}\right)$, da cui la tesi.

□

Esempio. La reciprocità quadratica permette di controllare velocemente se un elemento è un quadrato. Per esempio,

$$\left(\frac{101}{87}\right) = \left(\frac{14}{87}\right) = \left(\frac{7}{87}\right)\left(\frac{2}{87}\right)$$

Dato che $87 \equiv -1 \pmod 8$, si ha $\left(\frac{2}{87}\right) = 1$. Dunque

$$\left(\frac{101}{87}\right) = \left(\frac{7}{87}\right) = -\left(\frac{87}{7}\right) = -\left(\frac{3}{7}\right) = \left(\frac{7}{3}\right) = \left(\frac{1}{3}\right) = 1$$

Dunque 101 è un quadrato modulo 87.

Il simbolo di Legendre si può estendere per moltiplicatività, dando vita al simbolo di Jacobi.

Definizione 2.7. Sia $r \in \mathbb{N}$ dispari e sia $s \in \mathbb{N}$. Consideriamo due fattorizzazioni $r = \prod r_i$ e $s = \prod s_j$. Definiamo il simbolo di Legendre come

$$\left(\frac{s}{r}\right) = \prod_{i,j}\left(\frac{s_j}{r_i}\right)$$

Anche per il simbolo di Jacobi vale la reciprocità quadratica, ossia

Proposizione 2.8. Siano r, s interi positivi dispari. Allora

$$\left(\frac{r}{s}\right) = (-1)^{\frac{r-1}{2}\frac{s-1}{2}}\left(\frac{s}{r}\right)$$

Purtroppo il simbolo di Jacobi è solo un test per vedere se un numero non è un quadrato; il fatto che $\left(\frac{n}{m}\right) = 1$ non implica infatti che n sia un quadrato modulo m.

2.3 Un test di primalità: Miller-Rabin

Occupiamoci ora di studiare un test di primalità probabilistico. Per questo, utilizzeremo i due seguenti fatti teorici:

Proposizione 2.9 (Fermat)**.** Per ogni $x \in \mathbb{F}_p$, $x^p \equiv x$.

Proposizione 2.10. Sia $n \in \mathbb{N}$ e sia $x \in \mathbb{Z}/n\mathbb{Z}$. Se $x^2 = 1$ e $x \neq \pm 1$, allora n è composto.

Sia allora $p \in \mathbb{N}$ e scegliamo in maniera casuale $a \in \{2, \dots p-2\}$. Raccogliamo la massima potenza di 2 che divide $p-1$ e dunque $p - 1 = 2^r \cdot s$. Allora

- Si calcola $\gcd(a, n)$. Se questo è $\neq 1$, p non è primo.
- Si calcola a^s. Se $x_1 = a^s \equiv 1 \pmod{p}$, scegliamo un altro a.
- Si calcola poi $x_i = x_{i-1}^2 \pmod{p}$ per $i = 1, \dots, r$. Se $x_{i+1} \equiv 1$ e $x_i \neq -1$, allora p non è primo e possiamo terminare l'algoritmo.
- Se $x_{r-1} \neq -1$, allora p è composto. Se invece $x_{r-1} = -1$, si cambia a.

Dunque:

Algoritmo 2.4 Miller-Rabin

1: $q = n - 1$
2: $t = 0$
3: **while** $q \equiv 0 \pmod{2}$ **do**
4: $\quad t = t + 1$
5: $\quad q = \frac{q}{2}$
6: **end while**
7: Scegliere in maniera casuale $a \in \{2, \dots, n-2\}$.
8: $s = \gcd(a, n)$.
9: **if** $s \neq 1$ **then**
10: $\quad$ **return** p non è primo.
11: **end if**
12: $e = 0$
13: $b = a^q$
14: **if** $b = 1$ **then**
15: $\quad$ **go to** 7
16: **end if**
17: **while** $(b \neq \pm 1)$ & $(e \leq t - 2)$ **do**
18: $\quad b = b^2$
19: $\quad e = e + 1$
20: **end while**
21: **if** $b \neq -1$ **then**
22: $\quad$ **return** p non è primo
23: **else**
24: $\quad$ **go to** 7
25: **end if**

Si può provare che, se n è composto, la probabilità di trovare un a che fornisca la risposta "probabile primo" è minore di $1/4$. Consideriamo infatti una fattorizzazione di $n = \prod_{i=1}^{k} p_i^{e_i}$. Senza perdità di generalità, possiamo supporre che n sia dispari. Allora

$$\mathbb{Z}/n\mathbb{Z}^* \simeq \prod_{i=1}^{k} \left(\mathbb{Z}/p_i^{e_i}\mathbb{Z}\right)^* \simeq \prod_{i=1}^{k} \left(\mathbb{Z}/p_i^{e_i-1}\mathbb{Z} \times \mathbb{Z}/(p_i-1)\mathbb{Z}\right)$$

Ci basta stimare la probabilità che preso un elemento casuale, le sue componenti abbiano ordine diverso. In questo caso, infatti, sapremmo stimare il numero di $a \leq n$ che forniscono un elemento che al quadrato faccia 1 ma che non sia -1. Questa è maggiorata dalla probabilità che un elemento di

$$\mathbb{Z}/p^{h-1}(p-1)\mathbb{Z} \times \mathbb{Z}/q^{k-1}(q-1)\mathbb{Z}$$

abbia componenti di ordine diverso. Dato che $(p, q) = 1$, basta contare le coppie

$$(a, b) \in \mathbb{Z}/(p-1)\mathbb{Z} \times \mathbb{Z}/(q-1)\mathbb{Z}$$

tali che $\text{ord}(a) = \text{ord}(b)$. Chiaramente il caso peggiore si ha quando $p-1 \mid q-1$ e dunque è sufficiente effettuare tale stima su $\mathbb{Z}/(p-1)\mathbb{Z} \times \mathbb{Z}/(p-1)\mathbb{Z}$, ossia fornire un bound sulla sommatoria

$$\sum_{d|p-1} \varphi(d)^2 \leq \varphi(p-1)(p-1) \leq \frac{(p-1)^2}{2}$$

Sia $k \geq 2$ tali che $q-1 = k(p-1)$. Allora $q = kp-k+1$, ossia $q \geq 2p-1$. Unendo tutto, la probabilità di individuare un elemento le cui componenti abbiano ordine diverso è stimabile con

$$\frac{\frac{(p-1)^2}{2}}{p(2p-1)} \leq \frac{(p-1)^2}{2p(2p-1)} \leq \frac{1}{4}$$

2.4 Algoritmi di fattorizzazione

In questa sezione, studiamo i principali metodi per la fattorizzazione di interi; abbiamo infatti visto come questi giochino un ruolo fondamentale in molti dei principali sistemi crittografici. Supporremo chiaramente che i numeri presi in considerazione siano molto grandi.

Rimozione dei fattori piccoli Un preprocessing necessario prima di ogni algoritmo di fattorizzazione è la rimozione dei primi piccoli. Si suppone quindi di avere una tavola dei numeri primi piccoli (per esempio più piccoli di 500000). Si prova quindi a dividere n per i primi della tabella. A livello implementativo, conviene tenere in memoria non i primi stessi, ma la metà della differenza tra due primi successivi. Probabilisticamente, un numero casuale avrà molti fattori piccoli e quindi effettuare questo algoritmo banale permette di risparmiare molto tempo.

ρ di Pollard Sia $n \in \mathbb{N}$ e sia $f\colon \mathbb{Z}/n\mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}$ una funzione qualsiasi. Sia $a \in \{1, \dots, n-1\}$ e consideriamo la successione

$$\begin{cases} x_0 = a \\ x_{l+1} = f(x_l) \end{cases}$$

Per il lemma dei piccioni, necessariamente esistono k, t interi positivi tale che $x_k = x_{k+t}$ e poi si ripete. Sia $p \in \mathbb{N}$ un divisore di n. Allora la successione modulo p ha la stessa proprietà di periodicità (la riduzione modulo p è un omomorfismo). Quando $x_k \equiv x_{k+s} \pmod{p}$ e $x_k \neq x_{k+s}$, allora $\gcd(x_{k+t} - x_k, n) > 1$ e dunque avremo trovato un fattore non banale di n. L'idea dell'algoritmo è quindi semplice; per trovare la coincidenza, basta calcolare due successioni: oltre a quella degli x_k, si usa $y_k = f(f(y_{k-1}))$ con lo stesso valore iniziale, ossia $y_k = x_{2k}$. Per trovare una ripetizione, basterà allora confrontare ad ogni passo x_i e y_i. Lo pseudocodice è allora il seguente:

Algoritmo 2.5 ρ di Pollard

```
Scegliere a ∈ {1, ..., n − 1}
if gcd(a, n) ≠ 1 then
    return gcd(a, n) è un fattore di n
end if
x = a
y = a
d = 1
while d = 1 do
    x = f(x)
    y = f(f(y))
    d = gcd(|x − y|, n)
end while
if d = n then
    return Fallimento
else
    return d è un fattore di n
end if
```

$p-1$ di Pollard

Definizione 2.11. Sia B un naturale. n si dice B liscio se i fattori primi di n sono $\leq B$. n si dice B-liscio per potenza se tutte le potenze dei primi che dividono n sono minori di B

Il nostro obiettivo è fattorizzare un n per cui esiste un fattore p tale che $p-1$ sia B-liscio per potenza. In realtà, la nozione di naturale B-liscio è molto importante nella costruzione di algoritmi di fattorizzazione e per questo ne è stata studiata a fondo la densità, come per esempio nel teorema di Canfield, Erdos e Pomerance.

L'idea dell'algoritmo è quella di sfruttare il teorema di Fermat; sia p un fattore primo di n e sia a un intero coprimo con n (se no calcoliamo il gcd e fattorizziamo n). Se $p-1$ è un naturale B-liscio per potenze, allora detto

$q = \text{lcm}(1, 2, \ldots, B)$, si ha che $p - 1 \mid q$ e dunque

$$a^q \equiv 1 \pmod{p}.$$

Di conseguenza $\gcd(n, a^q - 1) > 1$ è un fattore di n (ma potrebbe essere n stesso). Diamo ora lo pseudocodice dell'algoritmo. Supporremo di avere in input anche la lista dei primi $p_1, \ldots, p_k$ minori di B, e poniamo $a = 2$.

Algoritmo 2.6 $p - 1$ di Pollard

```
1: x = 2, i = 0                                  ▷ Inizializzazione
2: while i < k do                                ▷ Costruzione di lcm(1, ..., B)
3:     i = i + 1
4:     l = ⌊log_{p_i} B⌋
5:     x = x^{p_i^l} (mod n)
6:     g = (x − 1, n)
7:     if g ≠ 1 then
8:         i = k
9:     end if
10: end while
11: if g ≠ 1 and g ≠ n then
12:     return g è un fattore di n
13: else
14:     return Fallimento
15: end if
```

Crivello Quadratico L'idea del crivello quadratico è molto semplice: supponiamo di trovare due interi distinti $a, b \in \mathbb{N}$ tali che $a^2 \equiv b^2 \pmod{n}$. Questo significa che $n \mid a^2 - b^2 = (a + b)(a - b)$ e dunque si può calcolare il massimo comune divisore tra n e i fattori in modo da trovare un fattore di n, ma può essere n stesso. Chiaramente, il problema implementativo di questo algoritmo è determinare un metodo efficiente per trovare a e b. Per questo, cerchiamo una lista di numeri $a \leq \sqrt{2n}$ per i quali $a^2 - n$ sia prodotto di numeri primi piccoli, ossia sia B-liscio per un certo B.

Consideriamo la funzione $f(x) = x^2 - n$ e, partendo da $a = \lfloor \sqrt{n} \rfloor + 1$, scriviamo la lista

$$f(a) \qquad f(a+1) \qquad f(a+2) \qquad f(a+3) \qquad \cdots \qquad f(b)$$

Sia ora p un primo (dispari) minore del bound B scelto. Se $t^2 \equiv n \pmod{p}$ non ha soluzioni, passiamo al primo successivo; altrimenti esistono due soluzioni α, β e dunque $f(\alpha + kp)$ e $f(\beta + kp)$ sono multipli di p per ogni k intero. Aggiorniamo allora la lista dividendo per p tutti gli $f(s)$ con $a \leq s \leq b$ tali che $s \equiv \alpha \pmod{p}$ o $s \equiv \beta \pmod{p}$, partendo dal minimo e saltando di p. A questo punto, se $p^2 \leq B$, si considera l'equazione

$$t^2 \equiv n \pmod{p^2}$$

Se questa ha soluzioni nell'intervallo $[a, b]$, allora si ripete il procedimento effettuato per p, ossia si dividono per p gli elementi della lista a partire dalle soluzioni, e si salta di p^2. Dunque si prova per p^3 (se $p^3 \leq B$) e così via. Se

invece non ha soluzioni, si passa al primo successivo. Rimane da esaminare il caso $p = 2$; questo in realtà non presenta differenze con il caso precedente, ma bisogna notare che l'equazione potrebbe avere un numero diverso di soluzioni dal caso precedente.

Alla fine di questo procedimento, nella tabella vi saranno alcune entrate uguali a 1 (altrimenti dovevamo prendere b o B più grandi). Si considerano allora i corrispondenti $t^2 - n$ e le fattorizzazioni di queste ottenute, creando una matrice in cui nel posto (i, j) compare l'esponente del j-esimo primo preso in considerazione che compare nella fattorizzazione dell'i-esimo elemento scelto con il crivello precedente (notiamo che la tabella poteva essere riempita mentre facevamo le operazioni sopra). Si cerca allora una combinazione lineare delle righe che sia nulla modulo due; da questa combinazione si ottiene un'uguaglianza di quadrati modulo n dal quale si può dedurre un fattore di n. Infatti otterremmo

$$m^2 = \prod f(a_i) = \prod (a_i^2 - n) \equiv \left(\prod a_i\right)^2 \pmod{n}$$

Questo è attualmente il secondo algoritmo più veloce conosciuto all'uomo dopo il GNFS per fattorizzare primi, con una complessità di

$$O\left(e^{(\log n)^{\frac{1}{2}} (\log\log n)^{\frac{1}{2}}}\right)$$

2.5 Algoritmi per il logaritmo discreto

Il problema che ci poniamo ora è quello del logaritmo discreto. Sia G un gruppo ciclico di ordine n, sia g un suo generatore e sia $a \in G$. Vogliamo determinare $s \in \mathbb{N}$ tale $g^s = a$. Per questo, esiste chiaramente l'algoritmo "brute force": calcolare $g^2, \ldots, g^i$ fino a quando $g^i = a$. Questo algoritmo è dell'ordine $O(n)$, ma esistono metodi più efficienti.

Baby-step/Giant-step Come sopra, consideriamo $B \in \mathbb{N}$ approssimazione della radice quadrata di n e calcoliamo

$$g \quad g^2 \quad g^3 \quad \ldots \quad g^B \quad g^{B+1}$$

A questo punto, calcoliamo $ag^{-i(B+1)}$ al variare di i, partendo da $i = 0$. Per un certo j, necessariamente $ag^{-j(B+1)}$ coinciderà con uno dei g^h con $h \leq B + 1$ e dunque basta confrontare i valori che si ottengono man mano con quelli calcolati all'inizio. Fino a qua non abbiamo ottenuto un miglioramento sensibile dell'algoritmo; l'idea è quella di ordinare i valori ottenuti all'inizio in modo da poter utilizzare la ricerca binaria per il confronto. In questo modo, la complessità dell'algoritmo è dominata dal costo di ordinamento delle prime B potenze. Otteniamo allora il seguente algoritmo, con complessità $O(\sqrt{n} \log \sqrt{n})$:

Algoritmo 2.7 Baby step-Giant step

1: Scegliere B che approssima $\sqrt{n}$.
2: Costruire il vettore $v = [g, g^2, \dots g^{B+1}] \pmod n$
3: Ordinare il vettore con l'algoritmo **QuickSort**
4: $l = \lceil \frac{n}{B} \rceil$
5: $i = 0$
6: $s = a$
7: **while** trovato = false **do**
8: $\quad s = sg^{-i(B+1)}$
9: $\quad$ Controllare, per ricerca binaria, se $s \in v$. Se vi appartiene, trovato = true e sia j l'indice corrispondente.
10: $\quad i = i + 1$
11: **end while**
12: **return** $j + (i-1)(B+1)$

ρ di Pollard per il logaritmo discreto
Consideriamo il gruppo $G = (\mathbb{Z}/n\mathbb{Z})^*$ e supponiamo di voler risolvere il problema del logaritmo discreto in base $b \in G$, ossia dato $a \in G$ trovare k tale che $b^k = a$. Supponiamo che

$$a^{r_1} b^{s_1} = a^{r_2} b^{s_2}$$

e sia $x \in \mathbb{N}$ tale che $b^x = a$. Allora, dall'equazione di sopra, otteniamo

$$xr_1 + s_1 \equiv xr_2 + s_2 \pmod{\text{ord}(b)} \tag{2.1}$$

Se trovassimo un'equazione di questo tipo, con $r_1 \neq r_2 \pmod{\text{ord}(b)}$, avremmo risolto il problema del logaritmo discreto. Suddividiamo il gruppo G in tre sottoinsiemi casuali G_1, G_2, G_3 in modo tale che $e \notin G_1$ e definiamo la funzione

$$\begin{array}{rccc} \varphi\colon & G & \longrightarrow & G \\ & g & \longmapsto & \begin{cases} g^2 \text{ se } g \in G_1 \\ g \cdot a \text{ se } g \in G_2 \\ g \cdot b \text{ se } g \in G_3 \end{cases} \end{array}$$

Consideriamo allora le successioni

$$x_0 = y_0 = e \qquad x_{i+1} = \varphi(x_i) \qquad y_{i+1} = \varphi(\varphi(y_i))$$

Dato che tutti gli elementi delle due successioni sono del tipo $a^r b^s$, basta allora trovare la prima coincidenza tra queste due successioni per ricondurci all'equazione (2.1). Il costo di questo algoritmo è dell'ordine di $O(\sqrt{n})$ dove n è la cardinalità di G, ma presuppone di conoscere in anticipo l'ordine di b, che può essere determinato per esempio con il Small Step-Giant Step.

Index Calculus L'algoritmo che presentiamo ora ha molte analogie con il crivello quadratico per la fattorizzazione degli interi e si basa anche questo sul fatto che esistono molti numeri lisci. Fissiamo allora un bound B e risolviamo il problema $g^x = a \pmod \ell$ per ogni primo $\ell \leq B$. A questo punto, si considera la successione di naturali

$$g^{-k} a \pmod p \quad k = 0, \dots, p-1$$

alla ricerca di un valore B-liscio. Otteniamo allora

$$g^{-k}a = \prod_{p_i \leq B} p_i^{e_i}$$

e dunque

$$-k + \log(a) = \sum_{p_i \leq B} e_i \log(p_i) \tag{2.2}$$

Se risolviamo allora il problema del logaritmo discreto per primi piccoli, sappiamo trovare $\log(a)$ dalla formula appena trovata. Consideriamo allora degli $0 < i < p$ casuali e calcoliamo $g_i = g^i$ alla ricerca di numeri B-lisci. Per gli indici i tali che g_i è B-liscio, otteniamo

$$g_i = \prod_{p_j \leq B} p_j^{e_i(j)}$$

ossia

$$i = \sum_{j \leq B} e_i(j) \log(p_j)$$

Se troviamo una sufficiente quantità di relazioni, otteniamo un sistema lineare che possiamo quindi risolvere in modo da trovare i logaritmi dei primi piccoli. Sotituendo nella relazione (2.2), otteniamo allora $\log(a)$, come voluto.

2.6 Curve ellittiche

Definizione 2.12. Una curva ellittica è una curva proiettiva liscia di genere 1.

Si può dimostrare che una tale curva è sempre piana (esiste un embedding in $\mathbb{P}^2$) e si può scrivere nella forma

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

con $4a^3 - 27b^2 \neq 0$ (a patto che la caratteristica del campo K sia diversa da 2 o 3). Tale forma si chiama forma normale di Weierstrass. Su una curva ellittica si può definire una struttura di gruppo. Per semplicità supponiamo che la curva abbia un flesso O all'infinito (quando è messa in forma di Weierstrass, questo è sempre vero), che faccia da elemento neutro per il gruppo. Dati due punti p, q della curva, definiamo $-(p+q)$ come il terzo punto di intersezione tra la curva e la retta passante per p e q (se $p = q$, allora si prende la tangente alla curva passante per p). Con questa operazione, la curva diventa un gruppo abeliano.

Abbiamo interesse a studiare le curve ellittiche dal punto di vista algoritmico; in particolare abbiamo bisogno di stimare il numero di punti di una curva ellittica su $\mathbb{F}_q$. Vale il seguente:

Teorema 2.13 (Hasse)**.** Sia C una curva ellittica su $\mathbb{F}_q$. Detto E il numero di punti razionali su C, vale

$$|E - (q+1)| \leq 2\sqrt{q}$$

Come ultimo fatto generale sulle curve ellittiche, esplicitiamo la somma nel gruppo dei punti razionali su una curva ellittica in forma di Weierstrass:

Proposizione 2.14. Sia C la curva ellittica di equazione $y^2 = x^3 + ax + b$ su un campo K di caratteristica diversa da $2, 3$. Siano $p_1 = (x_1, y_1)$ e $p_2 = (x_2, y_2)$ due punti non opposti di C e sia

$$m = \begin{cases} \frac{y_1 - y_2}{x_1 - x_2} & \text{se } p_1 \neq p_2 \\ \frac{3x_1^2 + a}{2y_1} & \text{se } p_1 = p_2 \end{cases}$$

Allora

$$p_1 \oplus p_2 = (-x_2 - x_1 + m^2, -y_1 + m(2x_1 + x_2 - m^2))$$

Lenstra ECM Utilizzando i risultati richiamati sulle curve ellittiche, è possibile costruire un algoritmo di fattorizzazione. Sia allora n un naturale. Se n fosse primo, allora $\mathbb{Z}/n\mathbb{Z}$ sarebbe un campo e dunque i punti di ogni curva ellittica formerebbero un gruppo. Per quanto visto nella proposizione, calcolare la somma di due punti richiede l'inversione di un elemento che in $\mathbb{Z}/n\mathbb{Z}$ potrebbe non esistere. Proprio questo è il punto dell'algoritmo; se n non è primo, spera nell'esistenza di un denominatore non invertibile nel calcolo della somma di punti della curva. Il fatto chiave dell'algoritmo è che, anche se su $\mathbb{Z}/n\mathbb{Z}$ l'equazione data non individua una curva ellittica (le curve ellittiche sono per definizione su un campo), abbiamo effettivamente una curva ellittica per ogni p primo che divide n (a patto che $\gcd(4a^3 + 27b^2, n) = 1$). Dato che il gruppo è finito, il punto scelto avrà ordine finito e sperabilmente diverso su ogni $\mathbb{Z}/p\mathbb{Z}$ con $p \mid n$. Quando $kP = O$ in un certo $\mathbb{Z}/p\mathbb{Z}$, ma non in $\mathbb{Z}/n\mathbb{Z}$, allora otterremo un denominatore non invertibile, come voluto. Si sceglie allora un bound B, nella speranza che la curva ellittica abbia un numero di punti B-liscio in $\mathbb{Z}/p\mathbb{Z}$ (e dunque lo sia anche l'ordine del punto P scelto) e si calcola kP, dove $k = \operatorname{lcm}(1, \ldots, B)$. L'unico problema rimane la scelta della curva ellittica e l'individuazione dell'inverso in $\mathbb{Z}/n\mathbb{Z}$. Dato che il problema di trovare un punto su una curva ellittica è un problema difficile, si sceglie la famiglia di curve

$$C_a \colon y^2 = x^3 + ax + 1$$

che contiene sempre il punto $(0, 1)$. Per quanto riguarda il calcolo dell'inverso, dato che a ogni passo si deve calcolare il gcd tra il denominatore dell'operazione e n, è conveniente usare l'algoritmo euclideo esteso per trovare i coefficienti di Bezout. Infatti, se y è il denominatore che compare

$$ay + bn = 1$$

allora a è l'inverso di y modulo n. Questa è la parte più costosa di ogni passo e dunque per rendere più efficace l'algoritmo si è pensato di "parallelizzare" il calcolo. Nella pratica, non si effettuano i calcoli su una sola curva ma si scelgono più curve ellittiche. Per calcolare l'inverso, si una allora il seguente algoritmo:

Algoritmo 2.8 Inversi paralleli

Input: $a_1, \ldots, a_k$ ▷ elementi di cui calcolare l'inverso
$c_1 = a_1$
for $i = 2, \ldots k$ **do**
    $c_i = c_{i-1} \cdot a_i \pmod{n}$
end for
Tramite l'algoritmo euclideo esteso, calcolare $uc_k + vn = d$
if $d \neq 1$ **then**
    if $d \neq n$ **then**
        return d è un fattore di n
    else
        $i = 1$
        while $d = 1$ **do** ▷ Cerchiamo un fattore proprio di n
            $d = \gcd(a_i, n)$
            $i = i + 1$
            return d è un fattore di n
        end while
    end if
else
    for $i = k, \ldots, 2$ **do**
        $b_i = uc_{i-1} \pmod{n}$
        $u = ua_i \pmod{n}$
    end for
    $b_1 = u$
    return $b_1, \ldots, b_k$ ▷ Gli inversi di $a_1, \ldots, a_k$
end if

Seguendo questo algoritmo, al posto di fare k algoritmi euclidei estesi, ne calcoliamo solo 1 ma richiediamo $2k$ moltiplicazioni, che generalmente sono meno costose.

Commentiamo ora informalmente la scelta del bound B. Data una curva ellittica su $\mathbb{F}_p$, per il teorema di Hasse il suo numero di punti è nell'ordine di p. Per il teorema di Canfield-Pomerance-Erdos, la probabilità che la cardinalità della curva ellittica su $\mathbb{F}_p$ con $p \mid n$ sia $L(p)^s$-liscia per potenze è dell'ordine di $L(p)^{-1/2s}$, dove $L(x) = e^{\sqrt{\ln(x) \ln \ln(x)}}$ e dunque conviene prendere un bound in quest'ordine di grandezza. Diamo ora lo pseudocodice dell'algoritmo; per semplicità, supponiamo di lavorare con una sola curva alla volta. Inoltre, assumiamo di conoscere i primi più piccoli del bound B scelto $p_1, \ldots, p_k$.

Algoritmo 2.9 ECM Lenstra

1: $E = (y^2 = x^3 + ax + 1)$ ▷ Scelta della curva ellittica
2: $x = (0, 1)$ ▷ Punto iniziale
3: $i = 0$
4: trovato = false
5: **while** $(i \leq k - 1)$ **and** (trovato = false) **do**
6: $i = i + 1$
7: $L = \lfloor \log_{p_i} B \rfloor$
8: $x = p_i^L \cdot x$ ▷ Multiplo nel gruppo dei punti
9: **if** l'operazione è andata a buon fine **then**
10: trovato = true
11: t è l'elemento non invertibile
12: **end if**
13: **end while**
14: **if** trovato=true **then**
15: $d = \gcd(n, t)$ ▷ Fattore di n
16: **return** d
17: **else**
18: **return** Fallimento ▷ Cambiare curva ellittica
19: **end if**

Questo è attualmente il terzo algoritmo più veloce conosciuto all'uomo per fattorizzare primi, con una complessità di

$$O\left(e^{\frac{1}{2}(\log n)^{\frac{\sqrt{2}}{2}}(\log\log n)^{1-\frac{\sqrt{2}}{2}}}\right)$$

2.7 Reticoli

Definizione 2.15. Un reticolo è un sottogruppo discreto di $\mathbb{R}^n$

Si può dimostrare che un reticolo è uno $\mathbb{Z}$-modulo libero di rango $\leq n$. I due problemi associati a un reticolo sono:

- SVP, "Shortest Vector Problem", ossia la ricerca del vettore non nullo del reticolo di minima norma
- CVP, "Closest Vector Problem", ossia la ricerca del vettore del reticolo a distanza minima da un vettore assegnato (non necessariamente del reticolo).

Tali problemi in generale sono NP-hard, ma diventano semplici se si dispone di una base ortogonale (o quasi). In questo caso, infatti, è sufficiente sfruttare l'ortogonalità per trovare la soluzione in maniera semplice. D'altronde, dato un reticolo, trovarne una base quasi-ortogonale è comunque un problema difficile.

Un'euristica per questi problemi proviene da particolari proprietà del reticolo. Dato un reticolo L e una sua base $v_1, \ldots, v_n$, possiamo considerare il *parallelogramma fondamentale*, ossia l'inviluppo convesso dei vettori del tipo $\sum e_i v_i$, con $e_i \in \{0, 1\}$. Questo non è univocamente determinato dal reticolo, in quanto dipende dalla base scelta, ma le matrici di cambiamento di base hanno

determinante 1 e dunque il volume del parallelogramma fondamentale è un'invariante del reticolo, indipendente dalla base. Nel caso in cui il reticolo abbia rango massimo, il volume del parallelogramma si ricava come determinante della matrice avente come righe i vettori della base. Se invece il reticolo ha rango k in $\mathbb{R}^n$ con $k < n$, detta B la matrice avente come righe la base del reticolo, è sufficiente calcolare la radice del determinante BB^t per ottenere il volume. Chiamato V questo volume, possiamo ricavare degli upper-bound per lo SVP. Infatti sappiamo che esiste un elemento del lattice v per cui

$$\|v\| \sim V^{1/n}\sqrt{n}$$

SVP Indaghiamo ora sull'esistenza di un algoritmo che trovi un'approssimazione del vettore più piccolo di un reticolo. Per questo, costruiamo una base ridotta o quasi ortogonale. Senza entrare nei dettagli, cerchiamo di spiegare come funziona. Consideriamo il caso in dimensione 2; supponiamo quindi di avere un reticolo $L \subseteq \mathbb{R}^2$ con generatori a, b. Se $a < b$, si modifica b sottraendogli un multiplo di a in modo da far diventare il risultato il più piccolo possibile Se dopo la modifica b è ancora maggiore di a, allora a è il vettore di minima lunghezza, altrimenti iteriamo. Si può mostrare che tale algoritmo è polinomiale. Questo purtroppo non si estende a dimensione maggiore di 2.

Per approssimare una base ridotta, si utilizza il procedimento di Gram-Schmidt come sotto-procedura e si cerca di approssimare la base ottenuta in questo modo. La prima idea che viene in mente è quella di copiare il metodo elaborato in dimensione 2: scegliere il vettore di norma minima, ridurre tutti gli altri e iterare. Questo algoritmo però rischia di ciclare.

LLL Reduction L'algoritmo LLL produce una base ridotta come nella seguente definizione:

Definizione 2.16. Sia $B = (b_0, \ldots, b_n)$ una base di un reticolo L e sia $B^* = (b_0^*, \ldots, b_n^*)$ la base ottenuta applicando l'algoritmo di Gram-Schmidt alla base B (senza normalizzazione). Consideriamo per $0 \leq j < i \leq n$ i coefficienti

$$\mu_{ij} = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle}$$

Diciamo che B è ridotta se

Size condition $|\mu_{ij}| \leq 1/2$

Lovasz condition Esiste $\delta \in (1/2, 1)$ tale che

$$\delta \left\|b_{i-1}^*\right\|^2 \leq \|b_i^*\|^2 + \mu_{i,i-1}^2 \left\|b_{i-1}^*\right\|^2$$

L'algoritmo prende allora in input una base B e la corrispondente base B^* e restituisce in output una base ridotta. L'algoritmo agirà tramite due operazioni:

- Si toglie a b_n una combinazione lineare degli elementi precedenti della base
- Si scambiano due elementi consecutivi della base.

Descriviamo più in particolare l'algoritmo.

Algoritmo 2.10 LLL reduction

```
1: k = 1
2: while k ≤ n do
3:     for j = k − 1, . . . , 0 do
4:         μ_{k,j} = ⟨b_k, b*_j⟩/⟨b*_j, b*_j⟩
5:         if |μ_{k,j}| > 1/2 then                ▷ Controllo Size Condition
6:             b_k = b_k − ⌊μ_{k,j}⌉ b_j
7:         end if
8:     end for
9:     μ_{k,k−1} = ⟨b_k, b*_{k−1}⟩/⟨b*_{k−1}, b*_{k−1}⟩
10:    if ⟨b*_k, b*_k⟩ ≥ (δ − μ²_{k,k−1})⟨b*_{k−1}, b*_{k−1}⟩ then ▷ Controllo Lovasz Condition
11:        k = k + 1
12:    else
13:        Scambiare b_k e b_{k−1}
14:        Aggiornare b*_k e b*_{k−1}
15:        k = max{k − 1, 1}
16:    end if
17: end while
```

Chiaramente, la parte dedicata al controllo della Size Condition fa in modo che la base in output rispetti tale condizione. Infatti, supponiamo che l'algoritmo trovi che $|\mu_{k,j}| > 1/2$ e che modifichi il corrispondente b_k con $\tilde{b}_k = b_k - \lfloor \mu_{kj} \rceil b_j$. Questa modifica comporta che, se $j \leq s < k$,

$$\tilde{\mu}_{k,s} = \frac{\langle b_k, b_s^* \rangle}{\langle b_s^* . b_s^* \rangle} - \lfloor \mu_{k,j} \rceil \frac{\langle b_j, b_s^* \rangle}{\langle b_s^*, b_s^* \rangle}$$

il secondo termine sia zero perché $b_s^* \perp \mathrm{Span}(b_1, \ldots, b_j)$ e dunque il procedimento lasci invariati i $\mu_{k,i}$ già esaminati.

Per quanto riguarda la condizione di Lovasz, questa nasce perché idealmente vorremmo che, detti L_i i reticoli generati dai primi i vettori di base, il volume del reticolo fondamentale di L_i sia minore di quello di L_{i+1}. La condizione di Lovasz determinerebbe proprio questo se $\delta = 1$ (e $\mu_{i,j} \equiv 0$?), ma in questo caso l'algoritmo potrebbe non terminare.

La base ridotta fornisce comunque una buona approssimazione in termini del vettore di minima lunghezza.

Proposizione 2.17. Sia L un reticolo di rango n, sia $\{v_1, \ldots, v_n\}$ una base ridotta ottenuta con $\delta = 3/4$ e sia $\{v_1^*, \ldots, v_n^*\}$ la corrispondente base di Gram-Schmidt. Allora

1. $\prod_{i=1}^{n} \|v_i\| \leq 2^{\frac{n(n-1)}{4}} \det L$
2. $\|v_j\| \leq 2^{\frac{i-1}{2}} \|v_i^*\|$ se $j \leq i$
3. $\|v_1\| \leq 2^{\frac{n-1}{4}} |\det L|^{\frac{1}{n}}$
4. $\|v_1\| \leq 2^{\frac{n-1}{2}} \min_{v \in L \setminus \{0\}} \|v\|$

Dimostrazione. La condizione di Lovasz implica che

$$\|v_i^*\|^2 \geq \left(\frac{3}{4} - \mu_{i,i-1}^2\right) \left\|v_{i-1}^*\right\|^2 \geq \frac{1}{2} \left\|v_{i-1}^*\right\|^2$$

ossia

$$\left\|v_j^*\right\|^2 \leq 2^{i-j} \|v_i^*\|^2$$

per ogni $j \leq i$. Utilizzando le formule di proiezione di Gram-Schmidt, otteniamo

$$\begin{aligned}
\|v_i\|^2 &= \left\|v_i^* + \sum_{j<i} \mu_{ij} v_j^*\right\|^2 \\
&= \|v_i^*\|^2 + \sum_{j<i} \mu_{ij}^2 \left\|v_j^*\right\|^2 \\
&\leq \|v_i^*\|^2 + \sum_{j<i} \frac{1}{4} \left\|v_j^*\right\|^2 \\
&\leq \|v_i^*\|^2 + \sum_{j<i} \frac{1}{4} 2^{i-j} \|v_i^*\|^2 \\
&\leq 2^{i-1} \|v_i^*\|^2
\end{aligned}$$

Da questo segue il punto 1, in quanto

$$\prod_{i=1}^{n} \|v_i\|^2 \leq \prod_{i=1}^{n} 2^{i-1} \|v_i^*\|^2 \leq 2^{\frac{n(n-1)}{2}} \prod_{i=1}^{n} \|v_i^*\|^2 = 2^{\frac{n(n-1)}{2}} (\det L)^2$$

Segue anche il punto 2, in quanto

$$\|v_j\|^2 \leq 2^{j-1} \left\|v_j^*\right\|^2 \leq 2^{j-1} 2^{i-j} \|v_i^*\|^2 = 2^{i-1} \|v_i^*\|^2$$

Inoltre,

$$\|v_1\|^n \leq \prod_{i=1}^{n} 2^{\frac{i-1}{2}} \|v_i^*\| = 2^{\frac{n(n-1)}{4}} \prod \|v_i^*\| = 2^{\frac{n(n-1)}{4}} |\det L|$$

Per l'ultimo punto, sia v un vettore non nullo del reticolo. Allora

$$v = \sum_{i=1}^{s} a_i v_i = \sum_{i=1}^{s} b_i v_i^*$$

Allora vale $a_s = b_s$ e dato che $a_s \neq 0$ (possiamo supporlo senza perdità di generalità), vale $|b_s| \geq 1$. Allora

$$\|v\|^2 = \sum_{i=1}^{s} b_i^2 \|v_i^*\|^2 \geq b_s^2 \|v_s^*\|^2 \geq \|v_s^*\|^2 \geq 2^{1-s} \|v_1^*\|^2 \geq 2^{1-n} \|v_1\|^2$$

da cui la tesi. □

In generale, se lasciamo δ incognito, otteniamo

$$\left(\frac{4}{4\delta - 1}\right)^{n-1} \|v\|^2 \geq \|v_1\|^2$$

Dato che $\delta \in (\frac{1}{2}, 1)$, la stima migliore che riusciamo a ricavare è con $\delta \sim 1$. Il risultato finale è che v_1 potrà essere più grande del vettore più piccolo del reticolo, ma per un coefficiente $\{\frac{4}{3}\}^n$.

CVP Per risolvere il problema di CVP, se si dispone di una base quasi ortogonale $(v_1, \ldots, v_n)$, si usa solitamente l'euristica di Babai. Sia allora $v \in \mathbb{R}^n$; scriviamo allora v come combinazione lineare a coefficienti in $\mathbb{R}$ $v = \sum_i c_i v_i$. Si sceglie allora $\tilde{v} = \sum_i \lfloor c_i \rceil v_i$ e si prende questa come approssimazione.

Algoritmo 2.11 Algoritmo di Babai

1: $v_1, \ldots, v_n$ base del reticolo
2: Scrivere $v = \sum_{i=1}^{n} a_i v_i$
3: **return** $\sum_{i=1}^{n} \lfloor a_i \rceil v_i$

Esiste anche un'altro algoritmo molto semplice, sempre attribuito a Babai. Questo algoritmo determina ad ogni passo un'approssimazione della proiezione di v sull'iperpiano generato dai primi $n-1$ e itera. Lo pseudocodice è il seguente:

Algoritmo 2.12 Algoritmo dell'iperpiano più vicino di Babai

1: $v_1, \ldots, v_n$ base del reticolo
2: Calcolare la corrispondente base di Gram-Schmidt $v_1^*, \ldots, v_n^*$
3: $w = v$
4: **for** $i = n, \ldots, 1$ **do**
5: $\quad w = w - \lfloor \frac{\langle w, v_i^* \rangle}{\langle v_i^*, v_i^* \rangle} \rceil v_i$
6: **end for**
7: **return** $v - w$

Reticoli e fattorizzazione I reticoli intervengono anche nella fattorizzazione di polinomi a coefficienti interi, in particolare nella fase di ricombinazione dei fattori. Sia $f \in \mathbb{Z}[x]$ il polinomio che vogliamo fattorizzare; supponiamo di aver già calcolato il sollevamento henseliano dei fattori di f in $\mathbb{Z}/p\mathbb{Z}$ a $\mathbb{Z}/p^n\mathbb{Z}$ con p^n maggiore del bound sui coefficienti dei divisori di f. Detto g uno dei fattori di f in $\mathbb{Z}/p^n\mathbb{Z}[x]$, consideriamo il reticolo

$$L = \{h \in \mathbb{Z}[x] \ : \ g \mid h \pmod{p^n} \text{ e } \deg(h) \leq \deg(f)\}$$

Sappiamo che esiste un unico polinomio irriducibile g_0 a coefficienti interi divisore di f, che sia multiplo di g su $\mathbb{Z}/p^n\mathbb{Z}[x]$. L'idea è che il vettore più corto di questo reticolo coincide con g_0, e questo risolve la fase di ricombinazione.

Capitolo 3

Crittografia

In questo capitolo, parleremo di crittografia, ossia ci occuperemo di codificare un'informazione contro un avversario attivo. Nel nostro modello, vi saranno due personaggi, A e B, che chiameremo Alice e Bob, e supporremo che Bob voglia scrivere un messaggio ad Alice. Inoltre, il messaggio sarà pubblico, ossia chiunque può leggere tale messaggio. Chiaramente, per mantenere segretezza nella comunicazione, il messaggio sarà cifrato; si userà quindi un linguaggio che solo Alice e Bob sanno leggere. Una terza persona, chiamata Eve (da eavesdropper) cercherà dunque di decifrare il messaggio. Formalmente, consideriamo allora un alfabeto $\mathcal{A}$ e sia $\mathcal{M}$ l'insieme dei messaggi ammissibili. Indichiamo con C l'insieme dei messaggi cifrati. Con $\mathcal{K}$ indichiamo l'insieme delle chiavi e ogni $k \in K$ determina una bigezione $E_k \colon \mathcal{M} \to \mathcal{C}$, detta cifratura e l'inversa $D_k \colon \mathcal{C} \to \mathcal{M}$ detta decifratura.

Definizione 3.1. Definiamo crittosistema come il dato di $\mathcal{A}$, $\mathcal{M}$, $\mathcal{C}$, $\mathcal{K}$ e l'insieme delle chiavi di cifratura e decifratura.

I sistemi si dividono in simmetrici e asimmetrici. In quelli asimmetrici la chiave di cifratura è pubblica, ossia nota a tutti, mentre la chiave di decifratura è privata, e la conosce solo il ricevente. Lo schema che sintetizza questo tipo di sistemi è il seguente: supponiamo che Bob debba inviare un messaggio ad Alice. Allora

- Alice pubblica la chiave di cifratura ma tiene privata la chiave di decifratura
- Bob scrive il messaggio utilizzando la chiave di cifratura
- Alice decifra il messaggio con la chiave che solo lei conosce.

In quelli simmetrici, invece, entrambe le parti della comunicazione conoscono una chiave privata comune.

Vediamo ora un po' di esempi.

3.1 Protocolli di Cifratura

RSA Tra i principali crittosistemi attualmente in uso vi è l'RSA, che è un crittosistema a chiave pubblica e asimmetrico.

- Alice sceglie due primi p, q e considera $n = pq$. Sceglie allora $e \in \mathbb{Z}/n\mathbb{Z}$ tale che $(e, \varphi(n)) = 1$ (ossia e sia un elemento del gruppo moltiplicativo) e calcola l'inverso d di $e \pmod{\varphi(n)}$
- Alice pubblica allora la chiave pubblica (n, e) e tiene segreta la chiave privata (n, d) (i primi scelti non sono più necessari).
- Bob, che deve mandare il messaggio m, pubblica allora $m^e \pmod{n}$ e Alice, che conosce d, può ricostruire il messaggio calcolando $(m^e)^d \equiv m \pmod{n}$.

La sicurezza di questo sistema è allora correlata alla difficoltà di fattorizzare un naturale. Infatti, per trovare l'inverso di e, è necessario conoscere $\varphi(n)$ e questo, nel caso particolare di $n = pq$, è equivalente a conoscere p e q. Infatti, se si conosce $\varphi(n) = (p-1)(q-1) = pq - p - q + 1$, si calcola $n - \varphi(n) + 1 = p + q$, così p e q sono le soluzioni di

$$f(x) = x^2 - (p+q)x + n$$

Rabin Un altro crittosistema asimmetrico simile all'RSA, è quello di Rabin.

- Alice sceglie due primi $p, q \in \mathbb{N}$ tali che $p \equiv q \equiv 3 \pmod{4}$. Questi saranno la sua chiave privata. La scelta di $p, q \equiv 3 \pmod{4}$ in generale serve solo a rendere più economica l'estrazione della radice da parte di Alice, esistono varianti con primi generici.
- Allora Alice pubblica $n = pq$, che è la chiave pubblica.
- Bob, che vuole scrivere ad Alice il naturale $m \leq n$, invia allora il messaggio $c = m^2 \pmod{n}$.
- Alice può allora calcolare in maniera semplice le radici di m, e tra queste trovare il messaggio di Bob.

Nell'ultimo passaggio, Alice deve indovinare (tra 4 possibilità) il messaggio di Bob. Questo è un grosso svantaggio di questo crittosistema, che infatti non ha mai avuto successo in questa forma.

Presentiamo una variante che sopperisce a questa mancanza, basata sempre sulla capacità di Alice di estrarre facilmente la radice quadrata di un numero modulo n.

- Bob codifica il suo messaggio tramite una sequenza binaria $(m_1, \ldots, m_r)$. Sceglie casualmente $a_1 \pmod{n}$ e calcola la sequenza $a_{i+1} := a_i^2 \pmod{n}$ per $i = 1, \ldots, r$, quindi prende b_i per $i = 1, \ldots, r$ come la cifra binaria meno significativa del rappresentante di a_i scelto in $[0, n-1]$: la sequenza dei b_i è nota per essere pseudocasuale. Il messaggio inviato da Bob sarà dunque (c, a_{r+1}), dove $c = (m_1 + b_1, \ldots, m_r + b_r)$.
- Alice per decifrare il messaggio prende a_{r+1} e determina $a_r, a_{r-1}, \ldots, a_1 \pmod{n}$ in sequenza, estraendo le radici quadrate. Questo può farlo in modo univoco in quanto, dato che $p \equiv 3 \pmod{4}$, se $y = x^2 \pmod{n}$, allora esattamente uno tra x e $-x$ è a sua volta un residuo quadratico modulo n (e analogamente per q). Ottenuta la sequenza $a_1, \ldots, a_r$, è banale recuperare $(m_1, \ldots, m_r)$ da $(m_1 + b_1, \ldots, m_r + b_r)$.

In realtà, dato che (a_i) ha la brutta abitudine di essere periodica, conviene fare attenzione che il suo periodo non sia troppo corto.

Knapsack Un altro sistema crittografico asimmetrico ormai obsoleto utilizza il problema dello zaino come arteficio matematico di protezione. L'idea è quella di scegliere dei pesi $(m_1, \dots, m_k)$ supercrescenti, ossia in modo tale che

$$\sum_{i=1}^{s} m_i < m_{s+1} \quad \forall s < k$$

Disponendo di una tale successione, dato un intero $m = \sum_{i=1}^{k} e_i m_i$ con $e_i = 0, 1$, è possibile ricostruire in modo univoco gli e_i. Supponiamo allora che Bob voglia mandare un messaggio a Alice.

- Alice sceglie una sequenza supercrescente $m_1, \dots, m_k$ e $n \in \mathbb{N}$ tale che $\sum_i m_i < n$.
- Alice sceglie $d \in \mathbb{N}$ tale che $\gcd(n, d) = 1$, e pubblica la sequenza $(dm_1, \dots, dm_k) \pmod n$, che diviene la chiave pubblica. (di solito tale sequenza viene permutata, ma per semplicità supporremo che questo non avvenga)
- Bob, che vuole inviare un messaggio binario $(e_1, \dots, e_k)$, calcola ed invia $c = \sum e_i dm_i \pmod n$.
- Alice calcola allora l'inverso di d tramite l'algoritmo euclideo; il messaggio $d^{-1}c = \sum e_i m_i$ è allora ricostruibile in maniera univoca grazie alla supercrescenza della successione degli m_i.

Questo algoritmo è stato presto rotto e quindi è andato in disuso. Infatti, supponiamo che Eve voglia decrittare il messaggio S conoscendo $(dm_1, \dots, dm_k) \pmod n$. Eve costruisce allora la matrice

$$\left(\begin{array}{cccc|c} 2 & 0 & \cdots & 0 & dm_1 \\ 0 & 2 & \cdots & 0 & dm_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 2 & dm_k \\ \hline 1 & 1 & \cdots & 1 & S \end{array}\right)$$

Le righe di questa matrice generano un reticolo e, se $x = (x_1, \dots, x_k) \in (\mathbb{F}_2)^k$ è il messaggio che Bob ha mandato ad Alice, allora il vettore

$$v = (2x_1 - 1, 2x_2 - 1, \dots, 2x_k - 1, 0)$$

appartiene al reticolo. Questo vettore ha tutte le componenti $0, \pm 1$ e dunque la sua lunghezza è dell'ordine di $\sqrt{n}$, come quella del vettore più corto del reticolo. Dunque basta applicare l'algoritmo LLL al reticolo per ritrovare (probabilmente) v e dunque ricostruire il messaggio.

Protocollo Diffie-Hellman Sono diffusi anche crittosistemi simmetrici, in cui la comunicazione si basa sulla conoscenza della chiave privata da parte di entrambi gli interlocutori. In questo caso, la parte delicata della trasmissione risiede nello scambio delle chiavi. Per questo, si usa solitamente il protocollo di Diffie-Hellman, che prevede il seguente scambio di messaggi: scelto G un gruppo moltiplicativo, e g un suo elemento invertibile,

- Alice sceglie $a \in \mathbb{N}$ e manda g^a a Bob
- Bob sceglie $b \in \mathbb{N}$ e manda g^b a Alice
- Alice e Bob calcolano g^{ab}, che diventa la loro chiave privata.

In questo caso, Eve potrebbe conoscere solo g^a e g^b, senza essere in grado di conoscere g^{ab}. L'unico modo che ha Eve di conoscere g^{ab} è quello di trovare x o y tali che

$$g^x = g^a \qquad\qquad g^y = g^b,$$

cioè risolvere il problema del logaritmo discreto, che abbiamo visto essere computazionalmente difficile.

Un algoritmo di cifratura per gli elementi del gruppo G può quindi essere il seguente:

- Bob vuole mandare il messaggio $m \in G$, e lo cifra con la chiave privata, trasmettendo $g^{ab}m$
- Alice calcola $g^{-ab}g^{ab}m = m$, ottenendo il messaggio originale.

Esiste anche una versione asimmetrica di questo protocollo di cifratura, che andiamo a descrivere

Protocollo ElGamal Scegliamo come sopra G un gruppo moltiplicativo, e g un suo elemento invertibile, e mettiamo che Bob voglia mandare un messaggio ad Alice.

- Alice sceglie $a \in \mathbb{N}$ come chiave privata e manda g^a a Bob, in modo che la chiave pubblica sia (G, g, g^a)
- Bob sceglie $b \in \mathbb{N}$, un messaggio $m \in G$, calcola $(g^a)^b$ e manda $(g^b, g^{ab}m)$ ad Alice
- Alice calcola $(g^b)^a$, e $g^{-ab}g^{ab}m = m$, ottenendo il messaggio originale.

Osserviamo che i protocolli Diffie-Hellman e ElGamal, oltre alla loro naturale implementazione in $(\mathbb{Z}/p\mathbb{Z})^*$, possono essere applicati anche alle curve ellittiche.

Hidden Field Equation Un altro protocollo di cifratura si basa sulla difficoltà di risolvere un sistema polinomiale di equazioni in più variabili.

Dato un campo $\mathbb{F}_q$ e una sua estensione finita $\mathbb{F}_{q^n}$, prendiamo $P(x)$ un polinomio univariato in $\mathbb{F}_{q^n}[x]$. Questo può essere visto come un'applicazioone

$$P : \mathbb{F}_{q^n} \to \mathbb{F}_{q^n}$$

ed è facilmente invertibile: se y sta nell'immagine di P, allora fattorizzando il polinomio $P(x) - y$ e prendendone le radici, otteniamo la controimmagine di y (in generale avrà più di una soluzione).

Se vediamo $\mathbb{F}_{q^n}$ come uno spazio vettoriale su $\mathbb{F}_q$, allora l'applicazione P si rappresenta come una n-upla di polinomi multivariati

$$(p_1, \ldots, p_n) : \mathbb{F}_q^n \to \mathbb{F}_q^n$$

che è invece molto difficile da invertire: se $(y_1, \ldots, y_n)$ sta nell'immagine, dovremmo infatti fare una base di Gröbner di $(p_i(x) - y_i)$.

Il nome HFE deriva dal fatto che la chiave privata contiene, oltre al polinomio P, anche due trasformazioni lineari affini che modificano la struttura di $\mathbb{F}_{q^n}$ come $\mathbb{F}_q$-spazio vettoriale, rendendone impossibile l'identificazione. Vediamo il protocollo in dettaglio

- Alice sceglie i campi $\mathbb{F}_q$, $\mathbb{F}_{q^n}$, un polinomio $P(x) \in \mathbb{F}_{q^n}[x]$, e due trasformazioni lineari invertibili $S, T : \mathbb{F}_q^n \to \mathbb{F}_q^n$. La chiave privata sarà (S, P, T).
- Alice calcola $\widetilde{P} = TPS : \mathbb{F}_q^n \to \mathbb{F}_q^n$ in forma di polinomi multivariati $(p_1, \ldots, p_n)$, e li rende pubblici.
- Bob, che vuole mandare un messaggio $m \in \mathbb{F}_q^n$ ad Alice, calcola $M = (p_1(m), \ldots, p_n(m))$, e lo spedisce, insieme ad una costante di controllo r che "indica" quale delle soluzioni del sistema rappresenta il messaggio originale.
- Alice, ricevuto M, applica $\widetilde{P}^{-1}(M) = S^{-1}P^{-1}T^{-1}(M)$ ed usa r per capire quale radice di P deve considerare.

Spesso, si usa prendere P in modo che i p_i risultanti siano quadratici (notiamo che S e T non influiscono sul grado dei p_i), o anche in modo che P sia bigettiva.

Per esempio, prendiamo n dispari, q una potenza di 2, e $P(x) = x^{q+1}$. Questo è il prodotto di due applicazioni lineari su $\mathbb{F}_q$: $x \to x^q$ e $x \to x$; viste come applicazioni di $\mathbb{F}_q^n \to \mathbb{F}_q^n$, le applicazioni lineari sono date da polinomi lineari in n variabili, dunque $P(x)$ sarà rappresentato da polinomi quadratici. Invece, visto come applicazione $P : \mathbb{F}_{q^n} \to \mathbb{F}_{q^n}$, è biettiva, poichè $(q+1, q^n - 1) = 1$.

Trovare una soluzione, anche unica, di un sistema di equazioni quadratiche, è molto lungo. Questo è uno degli algoritmi candidati ad essere resistente agli attacchi quantistici, ma non è altrettanto resistente contro metodi standard.

Polly Cracker Dato che risolvere un sistema polinomiale è difficile, si è pensato di utilizzare questo problema come mezzo per la creazione di un crittosistema. In questo crittosistema, Alice sceglie un ideale $I \subseteq K[x_1, \ldots, x_n]$ e $\xi \in V(I)$. Alice pubblica allora I (chiave pubblica) e tiene per se la radice scelta (chiave privata). Bob, che vuole scrivere un messaggio $c \in K$ ad Alice, sceglie un polinomio $f \in I$ random e invia $f + c$. Alice, che conosce una radice del sistema, calcola $(f + c)(\xi) = c$ e ricava il messaggio. La sicurezza di un tale crittosistema risiede nel fatto che c non è altro che il resto di $f + c$ secondo la divisione per una qualsiasi base di Gröbner di I, e calcolare una base di Gröbner è un problema computazionalmente difficile. Il problema è però che trovare una base di Gröbner è asintoticamente difficile ma non sempre è così.

GGH Alcuni crittosistemi si basano sulle proprietà dei reticoli. Il primo che studiamo è detto GGH in onore dei creatori (Goldreich, Goldwasser, Halevi) e prevede la scelta da parte di Alice di una base quasi-ortogonale M di un reticolo L. Per far questo, Alice controlla che il determinante sia vicino al prodotto delle norme delle colonne. Sceglie poi una matrice U di cambio di base (ossia a determinante 1) e pubblica $M' = UM$, in modo tale che non sia semplice ricavare M da M'. Bob, che vuole mandare un messaggio m ad Alice, invia $s = m \cdot M' + e$, dove e è un errore piccolo. Alice, per decrittare, calcola allora il problema del CVP applicato a s in modo da ottenere $m \cdot M'$, in quanto tale vettore appartiene al codice e se l'errore e è sufficientemente piccolo risolve il problema. Alice calcola poi $(m \cdot M')M'^{-1} = m$ per ottenere il messaggio.

NTRU Un altro protocollo legato ai reticoli è il protocollo NTRU. L'algoritmo prevede che Alice scelga n, p, q, con p dispari piccolo (di solito 3) e q pari e di medie dimensioni (di solito ~ 256). L'ambiente di lavoro è l'anello di polinomi $R = \mathbb{Z}[x]/(x^n - 1)$, ma talvolta si ridurranno i coefficienti modulo p o q. Alice sceglie $f, g \in R$ con coefficienti $0, 1, -1$, con f invertibile modulo p e q (in particolare, è importante che f non abbia la stessa quantità di coefficienti 1 e -1, altrimenti $f(1) = 0$ e non sarà mai invertibile). Detti f_q e f_p gli inversi di f rispettivamente modulo q e modulo p, Alice rende pubblico $h = pgf_q \pmod q$. Bob codifica il messaggio m come un polinomio in R a coefficienti compresi tra $-(p-1)/2$ e $(p-1)/2$, e sceglie un polinomio casuale r con coefficienti $0, 1, -1$ per oscurare il messaggio, quindi pubblica $rh+m \pmod q$. Alice, per decrittare il messaggio, calcola

$$a := f(rh + m) \equiv rpg + fm \pmod q$$

e sceglie un rappresentante in modo tale che i coefficienti siano compresi tra $-(q-1)/2$ e $(q-1)/2$. A questo punto Alice calcola $f_p a \equiv m \pmod p$ e dunque ha ricostruito m.

In realtà, l'algoritmo può fallire nel caso i coefficienti di $rpg + fm$ non siano compresi tra $-(q-1)/2$ e $(q-1)/2$, ma il più delle volte questo non succede. Una tattica efficace in caso di fallimento è spesso quella di decentrare il rappresentante, prendendo i coefficienti tra $-(q-1)/2 + kq$ e $(q-1)/2 + kq$ per qualche k. Anche questo può fallire (si parlerà di gap failure), ma nella pratica è così raro da poter essere trascurato.

Questo algoritmo è uno dei più efficenti e l'attacco alle chiavi è equivalente alla risoluzione di un SVP su un reticolo. Consideriamo infatti la matrice $2n \times 2n$

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

e il reticolo da essa generata (le righe sono i generatori). Allora il vettore $(\alpha f, g)$ appartiene al reticolo, in quanto $h = f_q \cdot g \pmod q$. Con una opportuna scelta di

α, $(\alpha f, g)$ è un vettore corto del reticolo e quindi abbiamo ricondotto la rottura del crittosistema a un problema di SVP.

3.2 Algoritmi di Firma

Uno dei problemi della crittografia è anche quello della firma digitale, ossia la sicurezza che il mittente dei messaggi sia effettivamente quello voluto. In questo contesto, si suppone sempre che Samantha ("the signer") voglia firmare un documento D per Victor ("the verifier"). Samantha, per questo, produrrà un documento firmato $\tilde{D}$ mediante una chiave privata che solo lei possiede; la firma apportata al documento viene verificata da Victor mediante la chiave pubblica.

I crittosistemi sono usati anche per questo, permettendo ad una persona di "firmare" il messaggio in maniera da certificare la propria identità tramite un algoritmo che consente di poter allegare a un messaggio un'informazione extra che ne garantisca l'autenticità. La firma avrà una scadenza, in quanto nel tempo potrebbe essere copiata e quindi falsificata. Un buon algoritmo deve però permettere la possibilità di controfirmare un documento: qualcuno potrebbe infatti certificare che in una data successiva al messaggio questo era autentico e valido. Questo permette di estendere la scadenza di validità di una firma.

Protocollo di Fiat-Shamir Il protocollo di Fiat-Shamir è uno dei principali metodi utilizzati per questo. Si basa sul principio della zero-knowledge, ossia non vengono rese pubbliche informazioni segrete che potrebbero essere intercettate da Eve. Il protocollo prevede $n, m^2 \pmod n$ pubblici, e m noto solo a Samantha.

- Samantha sceglie r e calcola $r^2 = x \pmod n$ e lo pubblica
- Victor chiede a Samantha uno tra r e mr (in maniera casuale)
- Samantha calcola $y = r$ o $y = mr$ a seconda della richiesta di Victor
- Victor calcola y^2 e vede se coincide con x o m^2x (a seconda della richiesta)

Tale protocollo è efficace; supponiamo infatti che Eve voglia ingannare Victor e fingersi Samantha. Allora Eve, che conosce m^2 ma non n, sceglie se mandare a Victor

1. r^2 con r casuale
2. r^2m^2 con r casuale

Di conseguenza

- Se Victor chiede r, Eve può rispondere solo nel caso in cui abbia scelto la strategia 1.
- Se Victor chiede mr, Eve può rispondere solo nel caso in cui abbia scelto la strategia 2.

Dunque Eve ha possibilità di successo nel 50% dei casi; ripetendo la prova un po' di volte, Victor può allora essere sicuro che il suo interlocutore sia effettivamente Samantha.

Firma RSA Vediamo per esempio come funziona l'algoritmo di firma RSA. Supponiamo quindi che Samantha abbia una chiave privata (n, d) e che sia pubblica la chiave (n, e). Samantha scrive allora un messaggio cifrato m senza informazioni rilevanti, e lo manda insieme al messaggio originale a Victor; lui decritta m con la chiave pubblica e lo confronta col messaggio originale, e se i due sono uguali, riconosce che Samantha effettivamente conosce la sua chiave privata.

Funzioni Hash Costruiamo ora una funzione particolare: dati un insieme più grande (A) e uno più piccolo (B), voglio trovare un'applicazione $f : A \to B$ "iniettiva" con "immagine vuota". Cioè dato $a \in A$, deve essere computazionalmente impossibile trovare b tale che $f(a) = f(b)$ (iniettività), e dato $y \in B$, deve essere computazionalmente impossibile trovare $x \in A$ tale che $f(x) = y$ (immagine vuota). Inoltre richiedo che f sia facile da calcolare.

Un'applicazione di un oggetto del genere è ad esempio quella di allocare degli elementi in memoria: se ho da allocare un insieme di informazioni, ciascuna delle quali è un oggetto lungo, voglio metterla da qualche parte in modo poi da poterla ritrovare. Se ho una funzione di questo tipo, che mappi quest'informazione lunga in una molto più corta, posso allocare l'elemento lungo nella posizione contrassegnata dall'immagine della funzione, così è poi possibile andarlo a ritrovare in tempo esiguo (il calcolo di f deve essere facile).

In questo caso, si dice che f fa un riassunto del messaggio, e la chiameremo "funzione hash".

Riassunti e Firme Un'altra applicazione di un oggetto simile consiste nell'identificazione di una stringa adatta ad essere codificato per fare la parte di una "firma". Questo è un problema che interessa gli algoritmi di firma tramite codifica. Per esempio, dopo un po' di firme del tipo $(m, \varphi(m))$, nel caso in cui la cifratura usata sia moltiplicativa/additiva, Eve potrebbe prendere due o più firme precedenti per combinarle ed ottenere una nuova firma valida.

La funzione f sopra, potrebbe essere una soluzione a ciò, perchè permette di usare un riassunto del messaggio trasmesso per la firma, ossia trasmettere un messaggio m (crittografato in qualche modo) e usare $f(m)$ per determinare la firma, poichè è computazionalmente impossibile risalire ad m.

Non tutti i protocolli di tipo crittografico ammettono una variante che ci dia una firma. Se abbiamo RSA, dato $a \in \mathbb{Z}_n$, la cifratura si fa mandando a in a^c e b in b^d in modo tale che componendo le due si abbia l'identità. Se una delle due è pubblica e l'altra è privata, sia può usare lo stesso algoritmo soggiacente sia per fare la cifratura sia per fare la firma. Sempre tenendo conto che quello che si firma è il riassunto, cifratura e decifratura sono altrettanto sicuri. C'è un punto importante: se una cifratura (coppia esponente pubblico/esponente privato) viene utilizzata per la cifratura, non deve essere usata per la firma. Utilizzare lo stesso esponente per chiave pubblica sia per cifratura che per firma non è sicuro: sapendo un certo numero di messaggi di cifra e di firma, si possono ritrovare informazioni sulla chiave privata.

ElGamal Vediamo ora l'algoritmo di ElGamal per la firma digitale, standardizzato come *DSS* (Digital Signature Standard).

Dapprima Samantha e Victor scelgono una funzione hash h, che applicato ad un messaggio, ne fornisce un riassunto di lunghezza stabilita in modo tale che, per ogni valore, sia computazionalmente impossibile determinare un messaggio di cui è immagine.

Samantha sceglie un primo q compreso tra 2^{160} e 2^{161} (per il postulato di Bertrand ne esiste uno, e la stima sui numeri primi ci dice che ce ne sono molti) e un numero primo p della forma $rq + 1$ (ne esistono infiniti per teorema di Dirichlet). Dato che $q|p-1$, allora $\mathbb{Z}/p\mathbb{Z}$ ha un elemento di ordine q. Per trovarlo, applichiamo l'algoritmo

1. Scegliamo in maniera casuale un elemento $x \in \mathbb{Z}/p\mathbb{Z}$.
2. Se $x^r \not\equiv 1 \pmod{p}$, allora $\alpha = x^r$ è l'elemento cercato
3. Se $x^r \equiv 1 \pmod{p}$ torniamo al punto 1.

La probabilità di scegliere un elemento di ordine divisore di r è $r/rq = 1/q$ e dunque la probabilità di successo è $1 - 1/q$; dato che q è grande, in pochi tentativi si trova un tale elemento. Il protocollo sarà

- Alice sceglie p, q, α come descritto sopra, e un intero $a < q$. La chiave privata sarà (a, q).
- Calcola $y = \alpha^a \pmod{p}$, e trasmette come chiave pubblica (p, α, y).

Una persona che vuole mandare un messaggio m (criptato in qualche modo), e si vuole identificare come Samantha, deve ora dimostrare di sapere a. Lo fa tramite questo procedimento:

- Alice sceglie ora k naturale tale che $\gcd(k, p-1) = 1$, quindi calcola

  $$t = \alpha^k \pmod{p} \qquad s = k^{-1}(h(m) - at) \pmod{p-1}$$

  ed invia $(t, s, h(m))$. Notiamo che essendo $q = ord(\alpha)$ dispari, allora preso t positivo e minore di p, esso non può essere $p-1$, dunque può essere visto come un elemento modulo $p-1$.
- Victor, che vuole verificare l'identità di Samantha, controlla che $1 \leq t < p-1$, e calcola

  $$v_1 = y^t t^s \pmod{p} \qquad v_2 = \alpha^{h(m)} \pmod{p}$$

  Se $v_1 = v_2$, allora effettivamente è Samantha che ha scritto il messaggio.

Infatti $s = k^{-1}(h(m) - at) \pmod{p-1}$ da cui

$$h(m) = at + ks \pmod{p-1} \implies \alpha^{h(m)} = \alpha^{at+ks} = y^t t^s \pmod{p}$$

Notiamo che s può essere calcolato solo conoscendo a, ed inoltre, essendo k incognito, anche se Eve leggesse la firma, non riuscirebbe a ricavare a.

GGH L'algoritmo GGH può servire anche per la firma digitale. Questo si basa sul fatto che, dato un elemento qualunque, per trovare un elemento del reticolo abbastanza vicino serve la base ridotta. Samantha dunque sceglie una base quasi ortogonale privata M (chiave privata) e pubblica una base $M' = UM$ (chiave pubblica) in modo tale che non sia facile risalire a M da quest'ultima. Dunque, se Samantha vuole firmare un documento d, risolve allora il problema del CVP con d. La soluzione s è allora la firma del documento. Victor può controllare che quella sia effettivamente una soluzione verificando che s sia effettivamente vicino a d. Questo metodo non è ottimale, in quanto ad ogni firma Bob rivela informazioni sul reticolo che possono essere utilizzate da un intercettatore per decrittare i successivi messaggi.