Teoria dei Codici e Crittografia

Dario Balboni

1 Introduzione ai Codici

2 Codici Lineari

3 Codici Ciclici

4 Codici BCH e RS

5 Codici di Goppa

6 Introduzione alla Crittografia

7 Test di Primalità

8 Fattorizzazione su $\mathbb{Z}$

9 Problemi con il Logaritmo Discreto

In questa sezione g indica la base del logaritmo e b l'elemento da trovare tale che $a = g^b$. L'ordine del gruppo viene indicato con n.

9.1 Baby-Step Giant-Step

Detto $m = \lceil \sqrt{n} \rceil$ costruiamo una tabella di (j, g^j) per $j = 1, \dots, m$. A questo punto per calcolare il logaritmo discreto di a calcoliamo ag^{-im} per $i = 1, \dots, m$ e controlliamo se esso è uguale ad un qualche g^j. Se ciò succede abbiamo che $ag^{-im} = g^j$ e quindi $a = g^{j+im}$.

9.2 ρ di Pollard per il logaritmo discreto

Si basa su un metodo lepre-tartaruga: dividiamo G in tre insiemi G_0, G_1, G_2 tali che $1 \notin G_1$. Definiamo quindi $f(x) = \begin{cases} ax & \text{se } x \in G_0 \\ x^2 & \text{se } x \in G^1 \\ gx & \text{se } x \in G_2 \end{cases}$ che in un algoritmo vero scriveremmo come due successioni sugli esponenti di a e di g.

Se troviamo una collisione $a^\gamma g^\beta = a^{\gamma'} g^{\beta'}$ allora si ha $b = (\gamma - \gamma')^{-1}(\beta' - \beta) \mod n$.

9.3 Pohlig-Hellman

Particolarmente efficiente se l'ordine del gruppo si fattorizza con primi piccoli. Scriviamo $n = \prod_{i=1}^r p_i^{e_i}$ e $b = \log_g a$. Vogliamo prima di tutto determinare $b_i \equiv b \mod p_i^{e_i}$ per poi rimontare la soluzione con il teorema cinese del resto.

Ogni intero b_i viene ottenuto calcolando le cifre l_j per $j = 0, \ldots, e_i - 1$ della sua espansione \$p$_i$\$-aria nel seguente modo: al passo j (posto $q = p_i$ e $e = e_i$) si calcola $\gamma = g^{l_0 + l_1 q + \ldots + l_{j-1} q^{j-1}}$ e si nota che (scrivendo $b = b_i + kq^e$) $(g^{n/q^{j+1}})^{kq^e} = 1$. Da ciò segue che $\tilde{a} = (a\gamma^{-1})^{n/q^{j+1}} = \tilde{g}_j^l$ e quindi si può usare un altro algoritmo per calcolare $b_i = \log_{\tilde{g}} \tilde{a}$.

9.4 Basi di fattori

Si scelgono un piccolo numero di elementi "irriducibili", che vengono chiamati base di fattori. Ad esempio si possono prendere i primi piccoli $\mathcal{B} = \{p_1, \ldots, p_h\}$. A questo punto cerchiamo degli r_i tali che g^{r_i} si riesca a scrivere con elementi della base: $g^{r_i} = \prod_{j=1}^{h} p_j^{t_{ij}}$. In questo modo otteniamo delle relazioni lineari $r_i = \sum_{j=1}^{h} t_{ij} x_j$ dove le incognite x_j sono i logaritmi di p_j. Quando abbiamo abbastanza relazioni risolviamo il sistema lineare (ricordando di usare il teorema cinese per evitare di incappare in zero-divisori).

Noti gli x_j possiamo prendere una potenza a caso s e controllare se ag^s si può scrivere nella base di fattori $ag^s = \prod_j p_j^{t_j}$. Se sì possiamo ricavare $b = (\sum_j t_j x_j) - s$.

10 Principali crittosistemi a chiave pubblica

10.1 Diffie-Hellman

È più che altro un protocollo di scambio di chiavi.

- Alice e Bob scelgono di comune accordo un primo p e un generatore g di $\mathbb{Z}_p^*$.
- Alice sceglie un numero segreto a e Bob un numero segreto b.
- Alice invia a Bob $A = g^a \mod p$, Bob invia ad Alice $B = g^b \mod p$.
- Alice calcola $B^a = g^{ab} \mod p$ e Bob calcola $A^b = g^{ab} \mod p$.

In questo modi essi ottengono un segreto comune. Un eventuale terzo che potesse ascoltare la loro conversazione imparerebbe solo p, g, g^a e g^b ed avrebbe bisogno di un metodo efficiente per calcolare g^{ab} dati g^a e g^b, che al giorno d'oggi non è noto (ed il meglio che si possa fare è il logaritmo discreto di uno dei due).

10.2 Elgamal

Protocollo di cifratura asimmetrica.

- Alice sceglie un primo p ed un generatore $g \in \mathbb{Z}_p^*$. Successivamente sceglie a e calcola $A = g^a \mod p$. La chiave pubblica è (p, g, A) mentre quella privata è a.
- Bob che vuole mandare un messaggio m ad Alice, sceglie un intero b e calcola $B = g^b \mod p$. Quindi calcola la chiave di cifratura $K = A^b = g^{ab}$, cifra il messaggio calcolando $m' = Km$ ed invia ad alice (B, m').
- Per decifrare, Alice calcola la chiave K come B^a, quindi recupera il messaggio calcolando $K^{-1}m' = m$.

10.3 RSA

Protocollo di cifratura asimmetrica. Denotiamo nel seguito con $\phi(n)$ la funzione di Eulero di n.

- Alice genera la sua coppia di chiavi: sceglie opportunamente due numeri primi p e q e ne fa il prodotto $n = pq$. Calcola infine $\phi(n) = (p-1)(q-1)$ e sceglie un numero e tale che $1 < e < \phi(n)$ e $\gcd(e, \phi(n)) = 1$. La coppia (n, e) è la chiave pubblica di Alice. Infine essa calcola d tale che $de \equiv 1 \mod \phi(n)$, che le servirà per decifrare. La sua chiave privata è formata da (p, q, d).
- Per mandare un messaggio m (compreso tra 0 e n e coprimo con n) ad Alice, Bob calcola $c = m^e \mod n$ ed invia c ad Alice.
- Per recuperare il messaggio Alice deve solamente calcolare $c^d = m^{ed} = m \mod n$.

La sicurezza di RSA discende dal fatto che per trovare d è necessario conoscere $\phi(n)$, ma questo è dimostrabilmente tanto difficile quanto fattorizzare n. Questo sistema si basa quindi **effettivamente** sulla difficoltà della fattorizzazione.

11 Curve Ellittiche

11.1 Equazione di Weierstrass

Dato K un campo, un'equazione della forma $Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3$ dove $a_1, \ldots, a_6 \in K$ viene detta equazione di Weierstrass ed identifica una curva ellittica.

11.2 Legge di Gruppo

Si può definire una legge di gruppo sulle cubiche. Ne scriviamo solo le formule: per calcolare $R = (x_3, y_3)$ somma di $P = (x_1, y_1)$ e $Q = (x_2, y_2)$ si ha $\begin{cases} x_3 = m^3 + a_1m - a_2 - x_1 - x_2 \\ y_3 = -(m + a_1)x_3 - q - a_3 \end{cases}$ dove $y = mx + q$ è la retta passante per P e Q (tangente se $P = Q$) con $m = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{se } P \neq Q \\ \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3} & \text{se } P = Q \end{cases}$ con $q = y_1 - mx_1$.

11.3 Teorema di Hasse

Data E una curva ellittica definita su $\mathbb{F}_q$ si ha la stima $\left|E - (q+1)\right| \leq 2\sqrt{q}$

11.4 Contare il numero di punti

Prendiamo un punto P e ne calcoliamo l'ordine con un metodo lepre-tartaruga sulla successione $P_i = i \cdot P$. Speriamo di trovare, nell'intervallo fornito dal Teorema di Hasse, un solo multiplo dell'ordine trovato. Eventualmente possiamo calcolare più ordini e verificare che cadano nella stima di Hasse solo i multi dei loro mcm.

11.5 Problema del logaritmo discreto

Il problema del logaritmo discreto è definibile su una curva ellittica come: dati $P, Q \in E$ determinare in più piccolo $k \in \mathbb{Z}$ tale che $Q = k \cdot P$. Si può quindi adattare lo scambio Diffie-Hellman alle curve ellittiche.

11.6 Scegliere una curva ellittica

Un metodo per scegliere una curva ellittica contentente un punto P è: prima scegliere il punto $P = (x, y) \in (\mathbb{F}_q)^2$, scegliere $a \in \mathbb{F}_q$ e porre poi $b = y^2 - x^3 - ax$.

11.7 Goldwasser-Kilian

È un test di primalità simile al test di Pocklington. Sia n un intero positivo, $a, b \in \mathbb{Z}_n$ e sia $E = \{(x, y) \in (\mathbb{Z}_n)^2 \mid y^2 \equiv x^3 + ax + b \mod n\} \cup \{O\}$ dove O è un simbolo che denota il "punto all'infinito". Sia inoltre m un intero. Supponiamo che esista un primo q che divide m e tale che $q > (n^{1/4} + 1)^2$. Se esiste $P \in E$ tale che $m \cdot P = O$ e $\frac{m}{q}P \neq O$ allora n è primo.

Le formule per la somma di punti prevedono anche delle divisioni. L'algoritmo potrebbe quindi doversi fermare se non possiamo dividere, ma in questo caso avremmo trovato che n non è primo (e ne avremmo addirittura trovato un divisore).

11.8 Algoritmo di fattorizzazione di Lenstra

È un algoritmo di fattorizzazione, simile all'algoritmo $p-1$ di Pollard. Si basa sull'osservazione che il calcolo di $k \cdot P$ richiede la divisione tra classi di resto modulo n, che può essere compiuta con l'algoritmo euclideo esteso se $\gcd(n, v) = 1$. Se $\gcd(n, v) = n$ comunque non ci sono problemi perché l'algoritmo restituisce il punto all'infinito della cubica, mentre se $\gcd(n, v) \neq 1, n$ abbiamo trovato un divisore di n.

1. Scegliamo un'equazione del tipo $y^2 = x^3 + ax + b$ in $\mathbb{Z}_n$ ed un punto P
2. Calcoliamo $eP \in E$, dove e è prodotto di molti numeri piccoli (prodotto di potenze di primi piccoli, oppure $B!$ per qualche B piccolo. In questo modo si può calcolare efficientemente).
3. Si possono presentare tre eventualità:
   - Se siamo riusciti a compiere tutte le operazioni, proviamo qualche altra curva e/o punto di partenza
   - Se abbiamo trovato $k \cdot P = O$ in qualche fase, ricominciamo da capo (poiché O è elemento neutro non ci sposteremo da esso).
   - Se ad un certo punto abbiamo $\gcd(v, n) \neq 1, n$, abbiamo trovato un fattore non banale di n.

12 Altri crittosistemi

12.1 Crittosistema di Rabin

13 Lezioni del Maestro

Disclaimer: Le parole del Maestro sono a volte di difficile decifrazione, e comunque invitano sempre ad una riflessione personale piuttosto che ad un bieco nozionismo. Pertanto siete pregati di non prendere con assoluta certezza quanto scritto di seguito che serve principalmente ad ispirare delle piacevoli conversazioni con i vostri amici.

13.1 Note sull'effettiva calcolabilità (Nota di Redazione)

Per avere un'idea di quanto una cosa sia effettivamente realizzabile (ed un attacco crittografico portabile a termine) diamo un'idea delle dimensioni attuali di memoria e di capacità di calcolo: Attualmente un processore può spingersi a qualche GigaHertz di clock e quindi (stimando grezzamente che ogni ciclo di clock corrisponda ad una operazione) dato il numero di operazioni da effettuare si può dividere per 10^9 per ottenere approssimativamente il numero di secondi che occorrono (ricordiamo che in un anno ci sono circa $3 * 10^7$ secondi). Questa quantità va ovviamente divisa per il numero di processori che si hanno a disposizione per effettuare il calcolo (che in un medio-grosso datacenter possono arrivare a 1000 macchine

con una trentina di processori l'una). Inoltre le quantità di memoria disponibili (sempre per un medio-grosso datacenter) viaggiano, nella migliore delle ipotesi, sull'ordine delle centinaia di PetaByte, ovvero circa $2^57 \simeq 10^17$ byte.

Molte dei parametri crittografici utilizzati oggigiorno (ad esempio la lunghezza dei primi in RSA) hanno lunghezze dai 300 ai 2000 bit. Ad esempio se dovessimo risolvere il logaritmo discreto in un gruppo di ordine primo di 500 cifre binarie, utilizzando baby-steps giant-steps avremmo bisogno di circa (più o meno) 2^250 bit di memoria e 2^250 operazioni, assolutamente proibitivo. Infatti vengono solitamente presi in considerazione anche eventuali attacchi da parte di servizi segreti e simili che possono avere come budget a disposizione anche parecchi miliardi di dollari per un singolo attacco. Moltiplicando i precedenti valori per 10^30 ci si può però ritenere protetti anche da tali attacchi.

13.2 Assunzioni per la sicurezza in crittografia (modelli)

13.2.1 P $\neq$ NP

Si assume sempre che P $\neq$ NP, dove si suppone che i problemi in P siano quelli efficientemente risolubili, mentre quelli NP-hard o NP-completi siano impossibili da risolvere.

Alcuni problemi che si pensavano essere strettamente in NP si sono poi rivelati essere in P. Ad esempio PRIMES (problema decisionale: dato n naturale è primo?):

- Algoritmo Miller-Rabin $\implies$ BPP
- AKS (2009) $\implies$ P

13.2.2 Scenari per la cifratura

Vedere questa risposta di Stack Overflow per una spiegazione concisa e soddisfacente, della quale ciò che segue è una brutta copia.

1. Indistinguibilità Dati due oggetti di cui uno è la codifica di un messaggio e l'altro è una successione casuale di bit i due sono indistinguibili: non c'è un algoritmo che permetta di dire chi è l'uno e chi è l'altro.

   Questa nozione viene spesso considerata sotto ipotesi aggiuntive (CPA, CCA, CCA2) nel setting di un gioco tra un challenger ed un attaccante nel quale l'attaccante ha diritto a consultare alcuni oracoli e il suo scopo è di rompere il sistema crittografico. Denoteremo con λ il parametro di sicurezza del crittosistema, con $(K_E, K_D) = KG(\lambda)$ la procedura di generazione della coppia chiave pubblica (di cifratura) e chiave privata (di decifratura). Gli algoritmi di cifratura E e D si suppongono essere noti a tutte le parti (così come KG) ma possono essere non deterministici (nonostante ciò verranno scritti come funzioni). È garantito che si riesca sempre a decifrare un messaggio cifrato: $D(K_D, E(K_E, M)) = M$.

   Si ha indistinguibilità quando, nei protocolli sotto esposti, la probabilità dell'avversario di vincere il gioco è minore di $\frac{1}{2} + \varepsilon$ dove ε è una funzione negligibile nel parametro di sicurezza λ.

2. IND-CPA: Indistinguibilità sotto Chosen Plaintext Attack **Descrizione**: L'avversario genera due parole di eguale lunghezza. Il challenger decide, casualmente, di cifrarne uno dei due. L'avversario deve quindi indovinare quale dei due è stato cifrato.

   Algoritmo

   (a) Challenger: istanzia la coppia di chiavi $(K_E, K_D) = KG(\lambda)$.

   (b) Avversario: sceglie m_0, m_1 due messaggi della stessa lunghezza e li manda al challenger. Può compiere altre operazioni in tempo polinomiale che includano chiamate all'oracolo di cifratura $E(K_E, -)$.

(c) Challenger: sceglie $b \in \{0,1\}$ casualmente, calcola $C = E(K_E, m_b)$ e manda C all'avversario.

(d) Avversario: esegue altre operazioni in tempo polinomiale che includano chiamate all'oracolo di cifratura. Successivamente manda in output $g \in \{0,1\}$.

(e) Se $g = b$ l'avversario vince.

Osservazioni: Questo modello è troppo debole, perché assume una sola interazione tra l'avversario e il challenger.

3. IND-CCA: Indistinguibilità sotto Chosen Ciphertext Attack **Descrizione**: Lo scenario è come il precedente ma l'avversario può chiamare oracoli di cifratura o decifratura **prima** di spedire il messaggio.

   Algoritmo

   (a) Challenger: istanzia la coppia di chiavi $(K_E, K_D) = KG(\lambda)$.

   (b) Avversario: sceglie m_0, m_1 due messaggi della stessa lunghezza e compie operazioni in tempo polinomiale includendo chiamate agli oracoli di cifratura $E(K_E, -)$ e di decifratura $D(K_D, -)$. Successivamente spedisce entrambi i messaggi al challenger.

   (c) Challenger: sceglie $b \in \{0,1\}$ casualmente, calcola $C = E(K_E, m_b)$ e manda C all'avversario.

   (d) Avversario: esegue altre operazioni in tempo polinomiale **senza poter chiamare nuovamente gli oracoli**. Manda in output $g \in \{0,1\}$.

   (e) Se $g = b$ l'avversario vince.

   Osservazioni: Questo modello è più sicuro perché prevede la possibilità di interazioni ripetute. Ciò significa che la sicurezza non si indebolisce con il tempo.

4. IND-CCA2: Indistinguibilità sotto Adaptive Chosen Ciphertext Attack **Descrizione**: Oltre alle capacità in IND-CCA, all'avversario è concesso consultare gli oracoli dopo aver ricevuto C, ma non può spedire C stesso agli oracoli.

   Algoritmo: come sopra ma (d) viene sostituito dalla possibilità di eseguire operazioni in tempo polinomiale con chiamate ad entrambi gli oracoli esclusa la decifratura di C.

   Osservazioni: La necessità di IND-CCA2 suggerisce che la possibilità di utilizzare l'oracolo di decifratura dopo aver conosciuto il testo cifrato può dare parecchio vantaggio in alcuni schemi, visto che le richieste all'oracolo possono essere scelte in base allo specifico testo cifrato.

13.3 Possibili attacchi a Crittosistemi

13.3.1 Insicurezza di RSA

RSA come spiegato nei libri è insicuro e non soddisfa IND-CPA per via della parziale omomorficità: se so crittare a e b allora so anche crittare $a \cdot b$. Inoltre se a viene sempre cifrato nello stesso modo è possibile sapere se un messaggio cifrato contiene a oppure no. Per questo è necessario aggiungere del padding e qualche informazione casuale al messaggio trasmesso per evitare questo tipo di attacchi.

Nell'RSA standard lo zero e l'uno vengono sempre codificati come sé stessi e questa è un'altra debolezza.

Inoltre chiave pubblica e chiave privata **non sono simmetriche**: se l'esponente privato è piccolo ($< \sqrt{n}$) esso può essere riconosciuto facilmente (vedere a questo proposito l'attacco di Coppersmith).

13.3.2 Attacco di prossimità all'implementazione RSA con TCR

Supposizione: Chi decifra il messaggio (e quindi conosce p e q) potrebbe voler velocizzare i conti ed esponenziare il messaggio modulo i due primi per poi ricomporre il risultato con il Teorema Cinese.

Tipo di attacco: L'attacco è basato sulla prossimità al computer ricevente: vi è un microfono che ascolta il computer che fa i calcoli. Potendo scegliere il messaggio in chiaro (chosen plaintext) si riusciva a scoprire che bit ci fosse in una certa posizione ascoltando solo il rumore che fa il computer durante una decifrazione. Con pochi passaggi si riusciva a ricavare completamente la chiave privata (p e q).

Soluzione: Basta non usare il TCR. In questo modo chi ascolta può imparare n (che comunque già conosce) ma non p e q.

13.3.3 Attacchi algoritmici a scambi Diffie-Hellman

Alcuni metodi di rottura di Diffie-Hellman non sono completamente esponenziali: vanno come $O(2^{\sqrt{n}})$ o $O(2^{\sqrt[3]{n}})$ (General Number Field Sieve). Oltretutto esistono attacchi basati sui computer quantistici (Fattorizzazione di Shor) che possono rompere questi sistemi in tempo polinomiale.

13.4 Funzioni di Hashing

Vogliamo trasformare una stringa di lunghezza arbitraria in una stringa di lunghezza fissata (hash) in modo che sia difficilmente contraffattibile, ovvero che sia possibilmente iniettiva. Non essendo ciò possibile si chiede che possa resistere ad un preimage attack.

13.4.1 Preimage Attack

Sia h la funzione di hashing (nota) ed x un messaggio (non noto). Sapendo $h(x)$ deve essere computazionalmente impossibile trovare un messaggio x' tale che $h(x') = h(x)$.

13.4.2 Derivazione da un crittosistema

Si può derivare una funzione di hashing da un crittosistema seppur in maniera non efficiente: si divide il messaggio M a blocchi $b_0, \ldots, b_k$. L'algoritmo specifica un blocco di partenza c_0 fissato per tutti. Si procede ora induttivamente per ottenere c_{i+1} si usa b_i per cifrare c_i. L'hash cercato è quindi c_{k+1}.

13.5 Algoritmi di Firma

13.5.1 Derivazione da un crittosistema ed una funzione di hashing

Mostro di saper cifrare un hash derivato dal messaggio originario. In questo modo il ricevente (sotto opportune ipotesi di difficoltà di collisioni e di sicurezza del crittosistema) può aspettarsi che sia stato io a mandare il messaggio.

Attenzione che normalmente non si possono usare le stesse chiavi per cifratura e firma perché si indeboliscono a vicenda visto che la firma - concettualmente - equivale ad una decodifica di messaggi arbitrari.

13.6 Merkle-Hellman

Crittosistema basato su Subset Sum. Funzionamento: dato un insieme di numeri $a_1, \ldots, a_n \in \mathbb{N}$ e $c_1, \ldots, c_n \in \{0, 1\}$ codifico il messaggio $(c_i)_i$ inviando $A = \sum_i c_i a_i \in \mathbb{N}$. È dimostrato che dato $\{a_i\}_i$ e A, trovare c_i è un problema NP-hard (ciò non significa che una certa istanza non possa essere molto semplice da rompere). Inoltre, affinché esso possa essere utilizzato crittograficamente, è necessario che (avendo a disposizione dei dati in più) sia possibile decifrarlo rapidamente. Inoltre la soluzione deve essere unica.

Se gli a_i sono supercrescenti, ovvero $a_{i+1} > \sum_{k=1}^{i} a_k$, dato $A = \sum_i c_i a_i$ è molto semplice trovare i c_i. Idea: posso prendere m e $d < m$ scelto casualmente (ma vicino ad m per mascherare anche i numeri piccoli) e considerare $b_i = da_i \mod m$ e pubblicare come base $\{b_i\}_i$. Quando ricevo $\sum_i c_i b_i$ moltiplico per l'inverso di d ed ottengo $\sum_i c_i a_i \mod m$ da cui recupero il messaggio originario.

13.7 Reticoli Interi

Sono interessanti perché per ora sono gli unici tipi di crittosistemi classici che ancora resistono ai computer quantistici.

Determinante di una matrice quadrata: $\det A = \sqrt{|\det (A^t \cdot A)|}$.

13.7.1 Teorema di Minkowski

Sia S un insieme convesso, $S \subseteq \text{Span}_\Lambda$ e simmetrico ($x \in S - x \in S$). Se $\mu(S) > 2^n \cdot \det \Lambda$ allora $S \cap \Lambda$ è non vuoto e contiene un $x \neq 0$.

13.7.2 Shortest Vector Problem

Dato un reticolo trovare il vettore non nullo più corto.

13.7.3 Closes Vector Problem

Dato un reticolo ed un vettore si chiede di trovare il vettore del reticolo più vicino al vettore dato.

13.7.4 Basi Ridotte

Vorremmo avere una descrizione del nostro reticolo con basi fatte da vettori "corti". Data una coppia di vettori a e b in $\mathbb{R}^2$ consideriamo $a + b$ e $a - b$. Diciamo allora che una base di un reticolo in $\mathbb{R}^2$ è ridotta se $||a||, ||b|| \leq ||a + b||, ||a - b||$.

Nel caso una delle disuguaglianze non valga si può sostituire uno dei due vettori con quello più corto trovato.

Se siamo in dimensione 2, l'algoritmo termina sicuramente restituendo una base ridotta per il reticolo. Viene quindi data una definizione di base $\$\delta\$$-ridotta in dimensione arbitraria per permettere all'algoritmo LLL di terminare.

Una base $A = \{a_1, \ldots, a_n\}$ si dice $\$\delta\$$-ridotta (con $\frac{1}{4} < \delta < 1$) se valgono le condizioni:

1. Detta $B = \{b_1, \ldots, b_n\}$ la base ottenuta dal processo di ortonormalizzazione di Gram-Schmidt, e chiamati $\mu_{ij} = \frac{\langle a_i, b_j \rangle}{\langle b_j, b_j \rangle}$ per $j < i$ i coefficienti di approssimazione vale che $|\mu_{ij}| \leq \frac{1}{2}$

2. Per ogni coppia di vettori consecutivi vale $|b_i + \mu_{i,i-1} b_{i-1}|^2 \geq \delta |b_{i-1}|^2$.

13.7.5 Algoritmo LLL (Lenstra-Lenstra-Lovasz)

Permette di trovare una base $\$\delta\$$-ridotta di un reticolo qualunque, fissato δ a priori, in tempo polinomiale.

Funziona nel "modo ovvio":

1. Si controlla se tutte le condizioni sono soddisfatte, nel qual caso ci si ferma

2. Se $\exists i, j$ tale che $\mu_{ij} > \frac{1}{2}$ allora si "aggiorna" la coppia di vettori $a_i \leftarrow a_i - \lfloor \mu_{ij} \rceil a_j$ (e si ricomputano i coefficienti di GS)

3. Se $\$\langle \text{ b}_\text{k}, \text{ b}_\text{k} \rangle < (\delta \text{ - } \mu^2_{\text{k, k-1}}) \langle \text{ b}_\text{k-1}, \text{ b}_\text{k-1} \rangle$ allora si scambiano a_k e a_{k+1} (e si riaggiorna tutto).

La parte furba di tutto è mostrare che l'algoritmo termina in tempo polinomiale, ma questo l'hanno già fatto Lenstra, Lenstra e Lovasz. Si noti che l'algoritmo è ben definito anche per il caso $\delta = 1$, ma non è assicurato che termini in tempo polinomiale. In particolare esiste una costante effettiva c_1 tale che l'algoritmo restituisce come primo vettore della base un vettore a_1 vicino al vettore più corto del reticolo originale s in modo che valga $|a_1| \leq c_1|s|$.

Idea della terminazione: chiamiamo Δ_i il determinante del sottoreticolo generato dai primi i vettori della base. Tutte le trasformazioni di Gram-Schmidt approssimato preservano i determinanti, mentre essi decrescono quando si effettua uno scambio. Se si considera $\Delta = \prod_i \Delta_i$ questo è un numero naturale che decresce ad ogni "passo" dell'algoritmo, quindi esso deve terminare.

13.7.6 Fattorizzazione di Polinomi a coefficienti interi

1. Prima di LLL Si considera il polinomio modulo p e lo si fattorizza in $\mathbb{F}_p$. Si una quindi il Lemma di sollevamento di Hansel per ottenere delle radici sui p-adici. Ora o i coefficienti sono troppo grossi (vedere la stima di Mignotte) oppure abbiamo trovato un candidato fattore. Se il polinomio f è irriducibile invece non ci resta altra scelta che provare tutti i fattori in $\mathbb{F}_p$.

2. Dopo LLL Possiamo utilizzare LLL per far diventare polinomiale l'euristica:

   .....

13.7.7 Risoluzione di Closest Vector Problem

LLL permette anche di trovare dei vettori vicini ad uno dato (anche se non di risolvere closest vector). Un problema equivalente a CVP è quello di trovare la classe di equivalenza più piccola modulo il reticolo di un vettore dato.

Per fare ciò ci sono principalmente due algoritmi:

- **Round off**, che è veloce ma che genera cattive approssimazioni: dato v da approssimare, si calcola $v = \sum_i c_i v_i$ dove i v_i sono la base del reticolo ed i $c_i \in \mathbb{Q}$. Basta ora arrotondare i c_i a degli interi per trovare un vettore nel reticolo. Ovviamente non è detto che sia il più vicino ed anzi ci sono casi in cui si hanno dei bound pessimi (si pensi a dei reticoli molto schiacciati in una direzione e molto allungati nelle altre).

- **LLL con base aumentata**, considero i vettori $v_1, \ldots, v_n, v$ (che non sono necessariamente più una base) ed eseguo l'algoritmo LLL fino a quando esso non arriva alla fine, avendo così prodotto un vettore piccolo che è v con sottratti alcuni dei v_i.

13.8 Applicazioni Crittografiche dei Reticoli

13.8.1 Merkle-Hellman

Si può rompere Merkle-Hellman utilizzando LLL considerando la matrice opportuna codificandolo come problema di Shortest Vector. Con le stesse notazioni di sopra, la matrice da considerare è:

$$\left(\begin{array}{ccc|c} 2 & & & 2Na_1 \\ & \ddots & & \vdots \\ & & 2 & 2Na_n \\ 1 & \cdots & 1 & 2NA \end{array}\right)$$

dove N è scelto molto grande in modo che (visto che A si può rappresentare come somma di alcuni a_i) si ottenga come vettore più corto $(\pm 1, \ldots, \pm 1, 0)$ dove si ha -1 al posto i se $c_i = 1$ e 1 al posto i se $c_i = 0$.

È stato inoltre osservato che non c'è modo di aggiustare Merkle-Hellman per farlo resistere a questo tipo di attacchi.

13.8.2 Goldreich-Goldwasser-Halevi

È un crittosistema asimmetrico basato sui reticoli. L'idea è quella di prendere un reticolo dato da una matrice quadrata di interi. Il reticolo ha una base privata "buona" ed una base pubblica che è "cattiva".

Attenzione: quella sotto pare essere una versione interpretata dal Maestro nel corso di una divinazione. Per essere sicuri di riferirsi a GGH con persone terze posso suggerire la consultazione della pagina Wikipedia corrispondente.

Costruzione della chiave: Si sceglie un λ piccolo (es $\lambda = 3$) e si riporta sulla diagonale della matrice. A questo punto la matrice viene "abbellita" con alcuni elementi fuori dalla diagonale, sempre in modo che λ sia l'elemento prevalente (ovvero che la base ottenuta alla fine del procedimento sia $\$\delta\$$-ridotta). Una volta trovata la matrice "buona", la si "abbruttisce": ad esempio la si moltiplica per una matrice unimodulare (ovvero ad entrate intere e determinante ± 1) oppure la si porta in forma di Hermite (la quale sarebbe comunque calcolabile da chi riceve la matrice e quindi non dà alcuna informazione ulteriore).

Per cifrare si prende un vettore di zeri e uni e lo si riduce lungo la forma di Hermite. Essendo corto il vettore di zeri e uni è lo Shortest Vector associato alla sua cifratura e può quindi essere ricostruito da chi ha la base buona.

Problemi: La chiave pubblica risulta essere di dimensione n^2 ed anche il numero di calcoli da fare è elevato. Inoltre cercando di evitare alcuni possibili attacchi si vede che $n >> 1000$ e diventa quindi impensabile realizzarlo.

13.8.3 Fiat-Shamir

È un protocollo di autenticazione: voglio mostrare a qualcun'altro che conosco un quadrato modulo n senza svelarlo (dove $n = pq$ prodotto di due primi può essere scelto globalmente da una parte fidata). Si sceglie quindi casualmente un x e si calcola $s = x^2$. La propria chiave pubblica è s e la chiave privata è x.

Ora se Alice vuole identificarsi con Bob (che conosce solo (n, s)) può fare nel seguente modo:

- Alice sceglie casualmente un y, calcola y^2 e lo comunica a Bob
- Ora Bob conosce s e $m = y^2$ mentre Alice conosce x, y e quindi anche xy. Bob può scegliere se farsi svelare da Alice uno tra y oppure xy.

  In questo modo Bob non può conoscere x (perché lo ottiene eventualmente moltiplicato per un numero casuale) ma può scegliere se controllare una tra:

  - Il fatto che Alice stia effettivamente calcolando quadrati di numeri (ovvero verificare che $m = y^2$)
  - Il fatto che Alice davvero conosca x (verificando che $sm = (xy)^2$)

Visto che la scelta di Bob viene fatta dopo che Alice ha già mandato il valore, se essa non conoscesse una fattorizzazione di n avrebbe solo il 50% di probabilità di poter rispondere correttamente. Ripetendo il protocollo molte volte si può far diminuire la probabilità di poter barare. Inoltre si può adattare il protocollo per effettuare tutte le N challenges con due sole comunicazioni tra Alice e Bob.

Non è però un protocollo Zero-Knowledge poiché Bob può imparare che un certo elemento con simbolo di Jacobi uguale a 1 **è** un residuo quadratico modulo n (e quindi modulo entrambi i primi).

13.8.4 Feige-Fiat-Shamir

Basato sul precedente è un protocollo di autenticazione a Conoscenza Zero: si ha cioè che la persona contro la quale ci stiamo autenticando non guadagna nessuna informazione sui parametri segreti del crittosistema tranne quelle che aveva già in precedenza.

Attenzione: Per una volta, non fidatevi di Wikipedia, riporta il protocollo in una maniera sbagliata: riferitevi alla paper originale DOI 10.1007/BF02351717.

Un'entità pubblica di cui si ha fiducia pubblica $n = pq$ prodotto di due numeri primi congrui a 3 modulo 4. In questo modo -1 è un nonresiduo quadratico il cui simbolo di Jacobi è $+1$.

- Alice sceglie casualmente $x_1, \ldots, x_v$ e calcola (scegliendo i segni in maniera random ed indipendente) $s_i = \pm x_i^{-2}$. Gli s_i costituiscono la chiave pubblica (assieme ad n). Il motivo per cui si scelgono dei $\pm$ è che in questo modo gli s_i possono a priori essere un qualunque numero con simbolo di Jacobi $+1$.

- Per l'autenticazione Alice sceglie casualmente un intero r ed un segno $\varepsilon \in \{1, -1\}$ e computa $t = \varepsilon r^2 \mod n$ che manda a Bob.

- Bob sceglie casualmente dei numeri $a_1, \ldots, a_v$ dove $a_i = 0, 1$ e li manda ad Alice.

- Alice computa $y = r x_1^{a_1} \cdot \ldots \cdot x_v^{a_v} \mod n$ e lo manda a Bob.

- Bob controlla che valga $t = \pm y^2 s_1^{a_1} \cdot \ldots \cdot s_v^{a_v}$.

La procedura viene ripetuta con diversi r ed a_i fino a quando Bob non è soddisfatto del risultato. Da notare che la possibilità che Alice risponda giusto senza che conosca le radici è 2^{-v} per ogni richiesta.

13.8.5 Digital Signature Standard

È un protocollo di firma di documenti. Si stabiliscono due numeri primi p e q in modo che q sia di 160 bit e p sia di 512 bit tali che $p = kq + 1$. Questa richiesta si fa in modo che $p - 1$ abbia un primo grosso nella sua fattorizzazione (poiché il problema del logaritmo discreto si può scomporre sui fattori).

Inizializzazione del sistema: q viene generato in maniera random e controllato con Miller-Rabin, mentre p viene cercato all'interno della progressione aritmetica. Per un noto teorema, la densità di primi all'interno di una progressione aritmetica è la stessa che tra tutti i numeri naturali (in particolare nel nostro caso $\frac{1}{512}$ è primo). p e q sono pubblici e possono anche essere condivisi tra diversi utenti. Bisogna inoltre controllare che la fattorizzazione di $p - 1$ sia difficile.

Bisogna poi trovare un generatore del sottogruppo ciclico di ordine q: prediamo un elemento casuale g' e calcoliamo $(g')^k$, il quale ha ordine 1 o q. Se $g'^k = 1$ viene scartato, altrimenti si considera $g = g'$ (anch'esso può essere reso pubblico).

Chiave personale: viene scelto x casuale tra 1 e q e si calcola $y = g^x$: y è la chiave pubblica mentre x è quella privata.

Firma: Dato un documento ne calcolo un hash h che sia di circa 160 bit. $0 < h < q$ e si sceglie $w \in (0, q)$ random. Si calcola quindi $g^w \mod p$ (che è un altro generatore del sottogruppo generato da g) e si calcola $r = g^w \mod p \mod q$ (il valore principale si intende). Si cerca poi un s tale che $sw \equiv h + xr \mod q$ e la firma cercata è (r, s).

Verifica della Firma: Si calcolano $u_1 = s^{-1}h \mod q$ e $u_2 = s^{-1}r \mod q$. Si verifica poi se vale che $g^{u_1} y^{u_2} \mod p = r \mod q$.

13.8.6 Altro sui Reticoli

1. Cifratura Se abbiamo a disposizione un reticolo con base buona e base cattiva (come in GGH), e se il messaggio è sufficientemente piccolo esso può essere interpretato come rumore da aggiungere al reticolo. Risolvendo il problema di Closest Vector con la base ridotta si decifra il messaggio.

2. Firma Fare la firma con i reticoli è molto più difficile perché l'hash può andare a finire in un qualsiasi punto del parallelogramma fondamentale e quindi con un numero sufficientemente alto di firme si può ricavare la base buona del reticolo. Quindi il tipo naïve di firma è insicura.

13.8.7 NTRU

Definito in precedenza come sistema crittografico sui polinomi a coefficienti interi $\frac{\mathbb{Z}[x]}{(x^n-1)}$. Dove "$x^n - 1$ può essere sostituito un po' da qualunque cosa".

Setup delle chiavi: Si scelgono due numeri primi (p, q) dove p è un numero primo piccolo (ma basta anche un irriducibile dell'anello). Serve inoltre che n sia primo altrimenti si hanno delle debolezze nel crittosistema.

Si scelgono inoltre due polinomi $f, g \in \frac{\mathbb{Z}[x]}{(x^n-1)}$ che devono essere invertibili modulo q ed f anche invertibile modulo p. Questi due polinomi costituiscono la chiave privata (e supponiamo che siano polinomi con coefficienti in $\{-1, 0, 1\}$).

Calcoliamo ora $h = \frac{g}{f} \mod q$ (quindi è bene che anche g sia invertibile altrimenti si potrebbe carpire qualcosa osservando gli zeri di h). La chiave pubblica è data da h e ph.

Cifratura: Sia m un messaggio che è un polinomio piccolo, con coefficienti in $\{-1, 0, 1\}$ ed r un polinomio random (sempre con coefficienti in $\{-1, 0, 1\}$) scelti secondo una distribuzione prestabilita (solitamente gaussiana o uniforme). Per cifrare si spedisce $c = m + rh \mod q$ (h non ha coefficienti piccoli e quindi "oscura" il messaggio).

Decifratura: Si esegue $cf \mod q \equiv fm + rfh \equiv fm + rgp$ dove f, m, r, g sono a coefficienti piccoli e quindi possiamo "cancellare" l'operazione di modulo q ed avere un polinomio ben determinato. Questo polinomio viene ridotto $\mod p$ per eliminare il rumore e successivamente moltiplicato per f^{-1} per riottenere m.