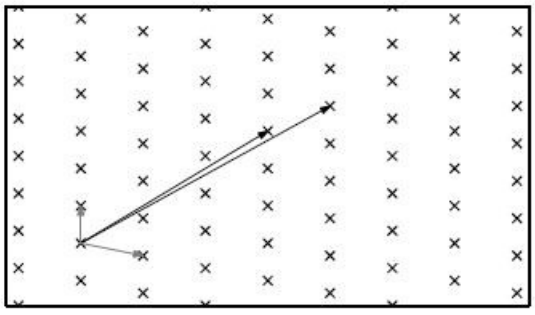
Reticoli

Dato una spazio vettoriale R^n , e una base $B=\{v_{1,}\ldots v_v\}$, il *reticolo intero* L generato da B è costituito da tutti i vettori che hanno coordinate intere rispetto a B :

$$L=\{k_1 v_1+\ldots+k_n v_n : k_{1,}\ldots k_n \, interi\}$$

Un reticolo ha molte basi diverse, con lo stesso numero di elementi, e ogni vettore del reticolo ha coordinate intere uniche rispetto a una data base. La lunghezza di un vettore è definita in modo analogo agli spazi euclidei.



Problema SVP (Shortest Vector Problem)

Consiste nel trovare un vettore in un reticolo, diverso da 0 e di lunghezza minima.

Per questo problema esiste un algoritmo polinomiale, chiamato LLL (Lenstra – Lenstra – Lovàsz) che permette di trovare un vettore la cui norma differisce da quella minima M per un fattore γ

$$\|v\| \leqslant \gamma M$$

L'algoritmo LLL parte da una base $B = \{v_{1,} \ldots v_v\}$ del reticolo e la trasforma in una *base ridotta* i cui vettori hanno lunghezza limitata; in particolare il primo vettore v della base ha lunghezza vicina a M.

Fino a quando la dimensione n del reticolo è piccola, il vettore v trovato è una buona approssimazione di un vettore di norma minima, ma il fattore di approssimazione γ cresce come $2^{O(n)}$, dove n è la dimensione del reticolo, e l'algoritmo diventa rapidamente inutilizzabile (anche se ha molte applicazioni in crittoanalisi).

Negli ultimi 20 anni sono stati proposti molti sistemi crittografici basati sui reticoli, soprattutto in due campi:

- Crittografia post-quantistica
- Crittografia omomorfa

Le ragioni sono diverse :

- Non esistono, ad oggi, algoritmi quantistici efficienti per i problemi sui reticoli, ad esempi SVP, mentre i problemi su cui si basano i sistemi di crittografia a chiave pubblica oggi in uso (fattorizzazione e logaritmo discreto) possono essere risolti in tempo polinomiale su un computer quantistico
- Nella crittografia omomorfa è importante che gli algoritmi di cifratura e decifrazione siano semplici (realizzabili con circuiti di complessità bassa). I sistemi sui reticoli utilizzano algoritmi lineari (spesso solo somme e riduzioni modulari), mentre ad esempio RSA richiede elevamenti a potenza con esponenti grandi.
- Gli algoritmi crittografici basati su SVP hanno una proprietà del tutto sconosciuta in altri campi, scoperta da Ajtai nel 1996: la sicurezza del sistema si basa sulla difficoltà del *caso peggiore* del problema soggiacente. In altre parole, se si ha successo nel forzare uno di questi sistemi (con probabilità non trascurabile) è possibile risolvere qualsiasi istanza del problema SVP (o di un altro problema simile).
- Questo non è vero nei sistemi di uso comune oggi, che si basano sulla difficoltà *media* di un problema: forzare RSA in un caso non permette di fattorizzare *qualsiasi* intero, ci sono casi, come abbiamo visto, in cui la fattorizzazione può essere relativamente facile.

Un esempio di sistema a chiave pubblica basato sulla difficoltà di SVP (Regev, 2004)

- N è un intero grande.
- La chiave privata h è un intero scelto in modo casuale nell'intervallo $[\sqrt{N}, 2\sqrt{N}]$.
- La chiave pubblica consiste in $m = O(\log(N))$ interi $a_{1,}\ldots, a_m$ in $\{0, 1, \ldots N-1\}$ prossimi a multipli interi di N/h, più un indice i_0 tale che a_{i_0} è prossimo a un multiplo *dispari* di N/h
- La cifratura avviene un bit alla volta: 0 viene cifrato come somma di un sottoinsieme casuale di $\{a_{1,}\ldots, a_m\}$ ridotta modulo N
- 1 viene cifrato nello stesso modo, ma prima di ridurre modulo N viene sommato $a_{i_0}/2$.
- Decifrazione: sia w un testo cifrato, e consideriamo il resto della divisione di w per N/h. Se è piccolo, w viene decifrato come 0, altrimenti come 1.

Il sistema è estremamente semplice: richiede solo somme e riduzioni modulari;

Ma è anche estremamente inefficiente, e difficilmente utilizzabile in pratica.

Nel sistema appena descritto non si fa nessun riferimento a reticoli, le operazioni di cifratura e decifrazione utilizzano solo l'aritmetica mod N; in che senso la sua sicurezza si basa sulla difficoltà di SVP ?

Attraverso due riduzioni:

- Una riduzione di (una variante) del problema SVP al problema di distinguere due distribuzioni sull'intervallo $[0, 1]$, una uniforme e l'altra centrata attorno ai multipli di N/h
- Una riduzione di quest'ultimo al problema di distinguere tra la cifratura di 0 e la cifratura di 1.

Così, se un avversario è in grado di distinguere tra la cifratura di 0 e quella di 1 con probabilità almeno ½ + ε, con ε non trascurabile, allora può distinguere tra le due distribuzioni, e risolvere il problema SVP per *qualsiasi* reticolo. In questo modo la sicurezza del sistema dipende dal *caso peggiore* di un problema matematico considerato intrattabile.

Rimane il problema dell'efficienza.

Sono stati proposti molti sistemi basati sui reticoli di efficienza maggiore. Uno di questi è NTRU, che usa l'aritmetica dei polinomi troncati di grado minore o uguale a un n fissato, ma non ha la proprietà della difficoltà del caso peggiore.

RSA è un omomorfismo moltiplicativo

Dati due testi cifrati con RSA:

$$C_1 = m_1^e \, mod \, N$$

$$C_2 = m_2^e \, mod \, N$$

$$C_1 C_2 = (m_1 m_2)^e \, mod \, N$$

La funzione di cifratura di RSA

$$E : Z_N \rightarrow Z_N \,, \quad E(x) = x^e \, mod \, N$$

è un omomorfismo di moltiplicazione. Per calcolare la cifratura del prodotto di due messaggi $m_{1,} m_2$, conoscendo i testi cifrati $C_{1,} C_2$, non è necessario conoscere la chiave segreta, basta moltiplicare tra loro i testi cifrati.

Questa proprietà di RSA da un lato permette la manipolazione dei messaggi (RSA è *malleabile*), dall'altro permette di eseguire calcoli direttamente su testi cifrati.

Il limite di questo schema è che permette di eseguire solo moltiplicazioni. Per poter calcolare funzioni qualsiasi su testi in chiaro, operando solo sui testi cifrati, occorre uno schema crittografico che permetta sia l'addizione che la moltiplicazione.

Schemi di cifratura “Fully Homomorphic”

Uno schema di cifratura *Fully Homomorphic* comprende, oltre alle funzioni di generazione delle chiavi, di cifratura E e decifrazione D, un algoritmo *Evaluate* che, per ogni coppia di chiavi (pk, sk), ogni funzione (calcolabile) F, e testi cifrati $C_i = E(m_i)$, restituisce $C = Evaluate(pk, C_1, \ldots, C_t)$ tale che

$$D(sk, C) = F(m_1, \ldots, m_t)$$

In altre parole, C è la cifratura di $F(m_1, \ldots, m_t)$, calcolata usando solo la chiave pubblica e i testi cifrati.

Le applicazioni di uno schema con questa proprietà sono innumerevoli (è stato chiamato il “*Sacro Graal*” della crittografia), e riguardano la possibilità di affidare non solo i dati, ma anche l'elaborazione degli stessi, a entità esterne inaffidabili, a cui non si possono affidare dati sensibili o riservati.

Il primo schema fully homomorphic è del 2009 (Gentry). Da allora sono stati proposti schemi più semplici e più efficienti, ma l'applicabilità pratica della crittografia omomorfa rimane ancora un problema.

Esempio: uno schema Fully Homomorphic a chiave privata

E' chiaro che gli schemi che conosciamo finora (RSA, ElGamal, AES) non hanno questa proprietà (al più ce l'hanno solo rispetto a *una* operazione aritmetica).

Per avere un'idea di come potrebbe funzionare uno schema del genere, descriviamo un esempio elementare, contenuto nella tesi di Gentry. E' uno schema a chiave privata, e usa solo interi[1]:

Fissato un intero N, la chiave è un intero *dispari* p > 2N (non necessariamente primo). La cifratura avviene un bit alla volta. Dato un bit b, il testo cifrato corrispondente è

$$c = b + 2x + kp$$

con x un intero casuale in (-N/2, N/2), e k un intero casuale (in un certo range).

Dato c, la decifrazione consiste semplicemente nel calcolare

$$(c \bmod p) \bmod 2$$

La parità di c è quella di k, quindi è casuale. Riducendo c mod 2 si ottiene b + k mod 2, che non dà informazioni su b. Per ricavare b è necessario eliminare prima il termine kp, e questo si può fare riducendo mod p (ma solo se si conosce p, che è segreto !).
b + 2x è nell'intervallo (-N, N), e per l'ipotesi p > 2N, si ha che $(-N, N) \subseteq (-p/2, p/2)$, perciò c mod p è esattamente l'intero b + 2x.

1 In questo contesto gli interi mod p vengono rappresentati con i numeri nell'intervallo (-p/2, p/2).

Il termine c mod p = b + 2x ha il significato di un “termine di rumore”, e la decifrazione avviene correttamente fino a quando questo termine è contenuto in (-p/2, p/2), altrimenti la riduzione mod p potrebbe alterare la parità di b + 2x.

Questo schema è omomorfo sia rispetto alla somma che alla moltiplicazione:

$c = c_1 + c_2 = b_1 + b_2 + 2(x_1 + x_2) + (k_1 + k_2)\,p$ è una cifratura di

$x_1 + x_2$

$c = c_1 * c_2 = b_1 * b_2 + 2(b_1 x_2 + b_2 x_1 + x_1 x_2) + kp$ è una cifratura di

$x_1 * x_2$

Il termine di rumore, però, aumenta in entrambi i casi. Perché la decifrazione sia corretta occorre che

$(b_1 + 2x_1) + (b_2 + 2\mathrm{x}_2) \in [-N\,,\,N]$
$(b_1 + 2x_1)(b_2 + 2\mathrm{x}_2) \in [-N\,,\,N]$

Questo schema, per quanto elementare, mostra un fenomeno che si ritrova in tutti i sistemi omomorfi: l'operazione di cifratura introduce un rumore intrinseco nel messaggio, e questo cresce quando vengono eseguite le operazioni omomorfe, in particolare la moltiplicazione, fino al punto che la decifrazione può fallire.

Bootstrapping

La prima soluzione al problema della crescita del rumore è stata proposta da Gentry (2009), mostrando anche la possibilità teorica della cifratura omomorfa. L'idea è che se uno schema è in grado di valutare omomorficamente (ossia su un testo cifrato) la sua stessa funzione di decifrazione, allora il rumore viene riportato al livello iniziale, come se si trattasse di un messaggio cifrato nuovo (che non ha subito operazioni omomorfe).

Supponiamo che C_1 sia una cifratura di P_1 con la chiave pubblica pk_1, e sk_1 la chiave privata corrispondente. Sia $\overline{sk_1}$ la cifratura di sk_1 con una seconda chiave pubblica pk_2.

Consideriamo il seguente algoritmo (semplificato rispetto all'originale, per leggibilità):

$$Recrypt(pk_{2,} D, \overline{sk_1}, C)$$
$$\overline{C_1} \leftarrow E(pk_{2,} C_1)$$
$$C_2 \leftarrow Evaluate(pk_{2,} D, \overline{sk_1}, \overline{C_1})$$

Come sopra, E e D sono le funzioni di cifratura e decifrazione. *Evaluate* valuta omomorficamente la funzione di decifrazione sul testo cifrato $\overline{C_1}$. Il risultato è una cifratura ex-novo di P_1 con una nuova chiave pubblica pk_2 in cui il rumore è riportato al valore iniziale.

Lo schema di Fan-Vercauteren e la libreria SEAL

Lo schema di Gentry non è praticamente applicabile per ragioni di efficienza.

Nel 2012 Fan e Vercauteren hanno presentato uno schema di "Somewhat Practical Fully Homomorphic Encryption".

Le primitive crittografiche di questo schema sono state implementate nella libreria SEAL (v2.0) (Simple Encrypted Arithmetic Library) che permette di testarne le possibilità applicative.

Nello schema FV i messaggi sono elementi dell'anello R_t dei polinomi troncati modulo un intero *t* (per i testi in chiaro), *q* (per i testi cifrati):

$$R_t = Z_t[x]/(x^n+1)$$
$$R_q = Z_q[x]/(x^n+1)$$

ossia i polinomi di grado minore di *n*, con coefficienti ridotti modulo *t* oppure *q,* con n una potenza di 2. Sia testi in chiaro che testi cifrati sono polinomi.

Gli interi *p, q* non sono necessariamente primi, e neppure coprimi.

Questi anelli hanno un'aritmetica completa, in cui la somma è per componenti (somma vettoriale) e il prodotto è il normale prodotto di polinomi, seguito dalla riduzione mod x^n+1 .

Osservazioni:

- I polinomi possono essere visti come vettori a coefficienti interi, quindi come elementi di un reticolo. Questo permette di ricondurre la sicurezza del sistema a problemi difficili sui reticoli.
- La cifratura non avviene più un bit alla volta, ma trasforma un vettore di interi.
- L'anello R_q possiede una struttura aritmetica, con somma e prodotto, che facilita le operazioni omomorfe.
- Per eseguire operazioni omomorfe su un testo cifrato non basta la chiave pubblica, occorre una *chiave di valutazione* **evk**, che *non* permette di decifrare il testo.

Lo schema FV contiene i seguenti algoritmi, dove λ è il parametro di sicurezza:

- SecretKeyGen(λ): genera la chiave segreta $sk = s \leftarrow R_2$
- PublicKeyGen(sk): genera la chiave pubblica $pk = ([-(as+e)]_q, a)$, dove $a \leftarrow R_q$, e un termine campionato secondo una distribuzione gaussiana fissata sugli interi.
- EvaluationKeyGen: genera $l+1$ chiavi di valutazione $evk = ([-(a_i s + e_i) + w^i s^2]_q, a_i)$, dove w è una base per gli interi.
- Encrypt(pk, m): posto pk = (p_0, p_1), calcola $ct = ([\Delta m + p_0 u + e_1]_q, [p_1 u + e_2]_q)$ con Δ il quoziente intero di q per t.

1. Decrypt(sk, ct): posto ct = (ct_0, ct_1), calcola $[\left\lfloor \frac{t}{q} [c_0 + c_1 s]_q \right\rceil]$

   Notare che la decifrazione consiste (a parte l'arrotondamento) nel valutare il polinomio $ct_0 + ct_1 x$ nella chiave segreta sk.

- Add($(ct_{0,} ct_1)$): calcola $(ct_0(0)+ct_1(0), ct_0(1)+ct_1(1))$

- Multiply($(ct_{0,} ct_1)$): $c_0=[\llbracket \frac{t}{q} ct_0(0) ct_1(0) \rrbracket]_q$

$$c_1=[\llbracket \frac{t}{q} ct_0(0) ct_1(1)+ct_0(1) ct_1(0) \rrbracket]_q$$

$$c_2=[\llbracket \frac{t}{q} ct_0(1) ct_1(1) \rrbracket]_q$$

I testi cifrati $(ct_{0,} ct_1)$ vengono moltiplicati come polinomi di primo grado:

$$(ct_0(0)+ct_0(1)x)(ct_1(0)+ct_1(1)x)$$

Il risultato è un polinomio di secondo grado, con c_2 coefficiente di x^2. Il passo successivo di *rilinearizzazione* trasforma il testo cifrato $c_0+c_1 x+c_2 x^2$ in un testo "standard" $(c_0^{'}, c_1^{'})$ con la stessa decifrazione:

Si esprime c_2 nella base intera *w* come $c_2=\sum_{i=0}^{l} c_2^{(i)} w^i$

$$c_0^{'}=c_0+\sum_{i=0}^{l} evk(i)(0) c_2^{(i)}$$

$$c_1^{'}=c_1+\sum_{i=0}^{l} evk(i)(1) c_2^{(i)}$$