Università degli Studi di Roma Tre

FACOLTÀ DI MATEMATICA

Appunti integrativi

Crittografia a chiave pubblica

CR410

Di:
Edoardo Signorini

A.A. 2016–2017, Semestre II

INDICE

1 INTRODUZIONE ALLA CRITTOGRAFIA 3
- 1.1 Cifrari storici 3
- 1.2 Crittosistemi 4
- 1.3 Crittoanalisi 5

2 CRITTOGRAFIA A CHIAVE PUBBLICA 6
- 2.1 Introduzione 6
- 2.2 Cenni di teoria della complessità 6
- 2.3 Calcolo della complessità di algoritmi noti 9
- 2.4 Problemi P e NP 11
- 2.5 Problema dello zaino 12
- 2.6 Crittosistema di Merkle-Hellman 13

3 IL CRITTOSISTEMA RSA 15
- 3.1 Introduzione 15
- 3.2 Descrizione del crittosistema 16
- 3.3 Test di primalità 18
  - 3.3.1 Test di Fermat 20
  - 3.3.2 Test di Solovay-Strassen 21
  - 3.3.3 Test di Miller-Rabin 29
- 3.4 Problemi di fattorizzazione 31
- 3.5 Attacchi ad RSA 34
  - 3.5.1 Attacco di Wiener 35
  - 3.5.2 Broadcast attack 38
- 3.6 Crittosistema di Rabin 39
- 3.7 Algoritmi di fattorizzazione 41
  - 3.7.1 Algoritmo p-1 di Pollard 41
  - 3.7.2 Algoritmo rho di Pollard 43
  - 3.7.3 Metodo di Fermat 44
  - 3.7.4 Metodo delle basi di fattorizzazione 45

4 IL PROBLEMA DEL LOGARITMO DISCRETO 48
- 4.1 Introduzione 48
- 4.2 Scambio della chiave di Diffie-Hellman 49
- 4.3 Cenni sui campi finiti 49
- 4.4 Alcune proprietà dei gruppi ciclici 50
- 4.5 Ricerca di radici primitive 51
- 4.6 Crittosistema di Elgamal 52
- 4.7 Sicurezza del logaritmo discreto 53
  - 4.7.1 Algoritmo di Shanks 54
  - 4.7.2 Algoritmo di Pohlig-Hellman 54
  - 4.7.3 Algoritmo Index-Calculus 56

5 FIRMA DIGITALE 59
- 5.1 Schemi di firma 59
- 5.2 Schema di Elgamal 61
- 5.3 Attacchi ad uno schema di firma 62

Indice analitico 62

1 | INTRODUZIONE ALLA CRITTOGRAFIA

In questo capitolo introduttivo daremo la definizione di crittosistema ed analizzeremo alcuni esempi classici di cifrari. Questo corso è incentrato sulla crittografia a chiave pubblica, la quale verrà introdotta nel successivo capitolo. Ciò che segue è quindi una breve introduzione alla crittografia classica che esula dagli scopi di questo corso.

Classicamente la crittografia nasce con lo scopo di nascondere il contenuto di un messaggio. Recentemente i suoi usi sono stati ampliati:

- Autenticazione di un messaggio o di un interlocutore.
- Scambio di una chiave segreta.
- Firma digitale.
- Condivisione di un segreto

Tradizionalmente si utilizzano dei personaggi fittizi per illustrare la situazione: classicamente avremo *Alice* che deve comunicare con *Bob*; entrambi devono però tenere conto della presenza di un attaccante *Eve* che può ascoltare la conversazione.

1.1 CIFRARI STORICI

Analizziamo di seguito due semplici cifrari usati in epoca antica:

Atbash è un cifrario a sostituzione monoalfabetica in cui la prima lettera dell'alfabeto è sostituita con l'ultima, la seconda con la penultima e così via. Ad esempio:

$$\text{ciao} \longrightarrow \text{YRZL}$$

Scitala è uno dei più antichi metodi di crittografia per trasposizione conosciuti: il meccanismo di codifica permetteva, nel caso la scitala fosse stata intercettata dal nemico, di mantenere segreto il contenuto del messaggio e, nello stesso tempo, consentiva al ricevente di verificarne l'autenticità, in quanto solo chi era dotato di una bacchetta identica a quella utilizzata dal mittente per preparare la scitala, poteva decifrare e leggere il messaggio. Nella figura figura 1.1 ve ne è una ricostruzione.

Figura 1.1: Una ricostruzione di scitala.

1.2 CRITTOSISTEMI

Definizione 1.1 – Crittosistema

Un *crittosistema* è una quintupla (P, C, K, E, D), dove

- P è un insieme finito di testi in chiaro o *plaintext*.
- C è un insieme finito di testi cifrati o *ciphertext*.
- K è un insieme finito di chiavi o *spazio delle chiavi*
- Per ogni $k \in K$ c'è una funzione di cifratura $e_k \in E$, con $e_k \colon P \to C$, e una funzione di decifratura $d_k \in D$, con $d_k \colon C \to P$, tali che

$$d_k(e_k(x)) = x$$

per ogni $x \in P$.

Osservazione. Chiaramente se $x, y \in P$ con $x \neq y$, si deve avere che, per ogni chiave k,

$$e_k(x) \neq e_k(y).$$

Ovvero le funzioni di E devono essere iniettve.

Esempio (Cifrario additivo)**.** Posti $P, C, K = \mathbb{Z}_{26}$, fissiamo $k \in [0, 25]$ e definiamo

$$e_k(x) = x + k \pmod{26} \qquad \text{e} \qquad d_k(y) = y - k \pmod{26}.$$

Quando $k = 3$ tale cifrario prende il nome di *cifrario di Cesare*.
Chiaramente possiamo riportarci all'alfabeto identificando $\mathbb{Z}_{26}$ con le lettere.

Affinché un crittosistema possa essere considerato efficiente e sicuro, si devono soddisfare due proprietà basilari:

- Dev'essere possibile calcolare ogni e_k e d_k in modo computazionalmente efficiente.
- *Eve* non deve essere in grado di risalire al testo in chiaro (o peggio, alla chiave) dal testo cifrato.

Nell'esempio dei cifrari additivi descritto poc'anzi, si hanno solamente 26 possibili chiavi. Ciò rende il crittosistema incredibilmente insicuro e attaccabile persino a mano.

Osserviamo che per definizione, P e C sono insiemi finiti e le funzioni di cifratura devono essere iniettive. Se in un crittosistema si ha $P = C$, sappiamo che una funzione $f \colon P \to C = P$ è iniettiva se e soltanto se è suriettiva. Quindi tutte le funzioni di cifratura sono biiettive. In tal caso E sono le permutazioni di P. Se P ha n elementi avremo che $E = S_n$ ha $n!$ elementi.

Esempio (Cifrari a sostituzioni)**.** Posti $P = C = \mathbb{Z}_{26}$ e $K = S_{26}$, per ogni $\pi \in K$ avremo

$$e_\pi(x) = \pi(x) \qquad \text{e} \qquad d_\pi(y) = \pi^{-1}(y).$$

In questo caso il numero di chiavi è pari a $26! \approx 4 \cdot 10^{26}$. Nonostante questo numero sia molto grande, ciò non basta a garantire la sicurezza di questo crittosistema. Questo metodo di cifratura lascia infatti inalterate le regolarità della lingua che permettono una facile decifratura.

1.3 CRITTOANALISI

Storicamente con il termine *crittonalisi*, si fa riferimento ai metodi di attacco contro un generico crittosistema.

Definizione 1.2 – Principio di Kerckhoffs

Il principio di Kerckhoffs afferma che la sicurezza di un crittosistema non deve dipendere dal tenere celato il crittoalgoritmo ma solo dal tenere celata la chiave.

In altre parole, il principio di Kerckhoffs afferma che nella costruzione di un crittosistema si deve lavorare affinché il sistema rimanga sicuro anche nell'ipotesi che l'attaccante conosca l'algoritmo di crittazione. Tale principio ha diversi vantaggi:

- É più facile tenere segreta la chiave.
- Se la sicurezza si basa sulla chiave, e la chiave viene scoperta, basta cambiare chiave.
- Si può usare lo stesso crittosistema per far comunicare diverse coppie di persone
- Un sistema che viene molto studiato (e attaccato) è più sicuro.
- Meglio che le debolezze, se ci sono, vengano scoperte e rese pubbliche.
- Se l'algoritmo è pubblico, non c''e rischio di reverse engineering.
- Si possono stabilire standard.

Oggi il principio di Kerckhoffs viene inteso in maniera più forte: l'algoritmo deve essere pubblico.

Tornando alla crittoanalisi, elenchiamo alcuni tipi di attacco:

Ciphertext only attack L'attaccante conosce una stringa y di testo cifrato. Cerca di risalire al testo in chiaro o alla chiave.

Known plaintext attack L'attaccante conosce una stringa x di testo in chiaro e il corrispondente testo cifrato y. Cerca di risalire alla chiave o di decrittare altri testi cifrati.

Chosen plaintext attack L'attaccante ha la possibilità di scegliere un testo in chiaro x e di ottenere il corrispondente testo cifrato y. Cerca di risalire alla chiave o di decrittare altri testi cifrati.

Chosen ciphertext attack L'attaccante ha la possibilità di scegliere un testo cifrato y e di ottenere il corrispondente testo in chiaro x. Cerca di risalire alla chiave.

2 | CRITTOGRAFIA A CHIAVE PUBBLICA

2.1 INTRODUZIONE

Nella crittografia simmetrica, di cui abbiamo dato una breve introduzione nel capitolo precedente, *Alice* e *Bob* condividono la stessa chiave k. La chiave è stata scelta e scambiata fra loro prima dell'inizio della comunicazione. Tramite la stessa chiave si ha accesso ad una funzione di cifratura e_k e ad una di decifratura d_k. Generalmente d_k è facilmente ricavabile da e_k; ciò significa che, tipicamente, se si è in grado di cifrare un messaggio allora lo si sa anche decifrare.

Come già accennato, prima di cominciare a comunicare è necessario scegliere e scambiarsi la chiave segreta tramite un canale sicuro. Se l'avversario intercetta la chiave, la comunicazione viene compromessa. L'idea della crittografia a chiave pubblica, o crittografia *asimmetrica*, è proprio quella di ovviare allo scambio della chiave.

Si cerca quindi di sviluppare un crittosistema in cui, data la funzione di cifratura e_k, sia computazionalmente difficile determinare d_k. Così facendo, *Bob* può rendere pubblica la sua funzione di cifratura e_k; tramite questa, *Alice* può scrivergli senza bisogno di accordi preliminari. *Bob* è inoltre l'unico che può decifrare il messaggio.

Concretamente, è necessario che la funzione di cifratura si una *funzione unidirezionale*. In seguito ne daremo una definizione precisa, informalmente una funzione invertibile $f\colon P \to C$, si dice unidirezionale se:

- Dato $x \in P$, il calcolo di $f(x)$ è "facile", cioè è realizzabile con una complessità polinomiale.
- Per quasi ogni valore di $y \in C$, il calcolo di $f^{-1}(y)$ è "difficile", cioè non è realizzabile con una complessità polinomiale.

$\varphi(n)$ *è la funzione di Eulero di* n

Esempio (Funzione unidirezionale)**.** Sia $n = p\,q$ con p, q numeri primi sufficientemente grandi e sia b un intero coprimo con $\varphi(n)$. Consideriamo la seguente funzione

$$f\colon \mathbb{Z}_n \longrightarrow \mathbb{Z}_n, x \longmapsto x^b \pmod n$$

f è una funzione ritenuta unidirezionale.

Sembra evidente che se la funzione è unidirezionale, anche per *Bob* è impossibile decifrare il messaggio. Proprio per questo motivo le funzioni usate nella crittografia a chiave pubblica sono le cosiddette *trapdoor one-way function*, ovvero funzioni unidirezionali che si invertono facilmente quando si conosce un'informazione supplementare. Tale informazione sarà tenuta segreta da *Bob*, il quale la utilizzerà per decifrare il messaggio. Riepilogando avremo

- Una *chiave pubblica*, resa nota a tutti, che verrà utilizzata per cifrare.
- Una *chiave privata*, nota solo a *Bob*, che verrà utilizzata per decifrare.

2.2 CENNI DI TEORIA DELLA COMPLESSITÀ

Ci occuperemo ora di stabilire, in maniera piuttosto grossolana, l'efficienza di un algoritmo. In questo corso ci occuperemo di algoritmi che operano sugli interi, pertanto l'input sarà costituito da uno o più interi.

L'efficienza di un algoritmo viene misurata nel tempo necessario a terminare con successo l'algoritmo stesso. Più direttamente, per non rendere la stima dipendente dal sistema usato per misurarla, il tempo dipenderà direttamente dalla lunghezza dell'input. Nello specifico, misureremo il tempo di esecuzione in base al numero di *operazioni bit elementari* necessarie a completare l'algoritmo.

Definizione 2.1 – Operazioni bit

Con *operazioni bit* intendiamo una fra le seguenti:

- Somma di due cifre binarie.
- Moltiplicazione di due cifre binarie.
- Divisione di un intero a due cifre binarie per una cifra binaria.
- Traslazione a sinistra o a destra di una stringa binaria.

Come già sottolineato, a noi interessa stimare approssimativamente il numero di tali operazioni all'interno di un algoritmo. Introduciamo ora la notazione $\mathcal{O}$ grande per formalizzare i concetti di complessità computazionale.

Definizione 2.2 – Notazione $\mathcal{O}$ grande

Siano $f, g\colon D \to \mathbb{R}^+, \mathbb{N} \subseteq D$. Si dice che g è *dominata* da f e si scrive $g \in \mathcal{O}(f)$, se esistono $k, N \in \mathbb{R}$ tali che

$$g(x) \leqslant k\, f(x),\ \forall\ x > N.$$

Notazione. Dalla definizione segue che $\mathcal{O}(f)$ è l'insieme delle funzioni dominate da f.

Osservazione. In generale se si ha

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = l \neq 0,$$

allora $f \in \mathcal{O}(g)$ e $g \in \mathcal{O}(f)$. D'altronde può accadere che quest'ultima condizione sia comunque verificata ma che il limite del rapporto non esista. Ad esempio, se $f(n) = \left(3 + (-1)^n\right)n^3$ e $g(n) = n^3$, chiaramente si ha $f \in \mathcal{O}(g)$ e $g \in \mathcal{O}(f)$, ma

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \left(3 + (-1)^n\right)$$

non esiste.

Esempio. Poniamo $D = \mathbb{N}$ e siano $f(n) = n^5, g(n) = 5n^5 + 3n + 2$. Chiaramente $f \in \mathcal{O}(g)$ poiché

$$f(n) \leqslant g(n),\ \forall\ n.$$

D'altronde avremo anche $g \in \mathcal{O}(f)$, infatti

$$g(n) = 5n^5 + 3n + 2 \leqslant 5n^5 + 3n^5 + 2n^5 = 10n^5 = 10\, f(n),\ \forall\ n.$$

Esempio. Sia $f(n) = n^3 \log_2 n$. Definitivamente si ha $\log_2 n \leqslant n$, da cui

$$f \in \mathcal{O}(n^4).$$

Proprietà 2.3. Siano $g_1 \in \mathcal{O}(f_1)$ e $g_2 \in \mathcal{O}(f_2)$, allora

$$g_1 + g_2 \in \mathcal{O}(f_1 + f_2).$$

Dimostrazione. Per definizione

$$g_1 \in \mathcal{O}(f_1) \implies \exists\, k_1, N_1 : g_1(x) \leqslant k_1 f_1(x),\ \forall\, x > N_1.$$

Analogamente

$$g_2 \in \mathcal{O}(f_2) \implies \exists\, k_2, N_2 : g_2(x) \leqslant k_2 f_2(x),\ \forall\, x > N_2.$$

Siano $N = \max\{N_1, N_2\}$ e $k = \max\{k_1, k_2\}$. Allora se $x > N$ avremo

$$g_1(x) + g_2(x) \leqslant k_1 f_1(x) + k_2 f_2(x) \leqslant k\big(f_1(x) + f_2(x)\big).$$

ovvero $g_1 + g_2 \in \mathcal{O}(f_1 + f_2)$. □

Proprietà 2.4. Siano $g_1 \in \mathcal{O}(f_1)$ e $g_2 \in \mathcal{O}(f_2)$, allora

$$g_1 g_2 \in \mathcal{O}(f_1 f_2).$$

Dimostrazione. Analoga alla precedente. □

Proprietà 2.5. Siano $g_1 \in \mathcal{O}(f_1)$ e $c > 0$, allora

$$c\, g_1 \in \mathcal{O}(f_1).$$

Dimostrazione. Analoga alla precedente. □

Definizione 2.6 – Grandezza di un intero

Definiamo la *grandezza* di un intero n come la sua lunghezza in notazione binaria $L_2(n)$.

Osservazione. Se $2^{k-1} \leqslant n < 2^k$ avremo

$$k - 1 \leqslant \log_2 n < k \implies k = L_2(n) = [\log_2 n] + 1,$$

per cui $L(n) \in \mathcal{O}(\log_2 n)$.

Esempio. Se $n = 24$ la sua grandezza è 5, infatti

$$24 = 11000_2$$

Se l'input dell'algoritmo A è costituito dagli interi $n_1, \dots, n_s$, poniamo

$$n = \max\{|n_1|, \dots, |n_s|\} \qquad \text{e} \qquad k = L(n).$$

Con $T(A)$ definiamo il tempo di esecuzione di A, pari a circa il numero di operazioni bit elementari che servono per l'esecuzione dell'algoritmo. $T(A)$ sarà pertanto una funzione di k. In particolare noi saremo interessati ad una stima del tipo $T(A) \in \mathcal{O}\big(f(k)\big)$.

Definizione 2.7 – Algoritmo polinomiale

Un algoritmo A si dice *polinomiale* se esiste $d \geqslant 1$ tale che

$$T(A) \in \mathcal{O}(k^d).$$

Notazione. L'esponente d si dice *ordine di* A.

Definizione 2.8 – Algoritmo esponenziale

Un algoritmo A si dice *esponenziale* se esiste $c > 0$ tale che

$$T(A) \in \mathcal{O}(2^{c\,k}).$$

Notazione. Diremo che A è *sottoesponenziale* se

$$T(A) \in \mathcal{O}(2^{f(k)}) \qquad \text{dove} \lim_{n\to\infty} \frac{f(n)}{n} = 0.$$

2.3 CALCOLO DELLA COMPLESSITÀ DI ALGORITMI NOTI

In questo paragrafo studieremo la complessità computazionale di alcuni algoritmi basilari e/o noti.

In generale faremo riferimento ad a e b come due interi con $a \geqslant b$, inoltre denoteremo

$$L(a) = k \qquad \text{e} \qquad L(b) = h.$$

Proposizione 2.9 – Complessità della somma

Consideriamo l'algoritmo per la somma di due interi $a + b$, allora

$$T(a + b) \in \mathcal{O}(k).$$

Dimostrazione. Una somma fra due interi è compiuta attraverso k somme elementari tra i bit che li compongono. Eventualmente tali somme potrebbero causare un riporto che necessita di un'ulteriore somma. Al più vi saranno quindi $2k$ operazioni bit. Ne segue

$$T(a + b) \in \mathcal{O}(k).$$ □

Osservazione. Analogamente si mostra $T(a - b)$.

Proposizione 2.10 – Complessità della moltiplicazione

Consideriamo l'algoritmo per il prodotto di due interi $a\,b$, allora

$$T(a\,b) \in \mathcal{O}(k^2).$$

Dimostrazione. Per ognuna delle h cifre di b bisogna eseguire k moltiplicazioni elementari tra bit, per un totale di $h\,k$ moltiplicazioni elementari. Infine è necessario compiere $h-1$ somme tra i prodotti parziali precedentemente calcolati. Per la proposizione precedente sappiamo che la somma ha una complessità lineare ed ha pertanto un ordine inferiore rispetto alle moltiplicazioni precedenti. Quindi

$$T(a\,b) \in \mathcal{O}(h\,k) \subseteq \mathcal{O}(k^2).$$

□

Osservazione. Esistono altri algoritmi che eseguono il prodotto fra interi in modo più efficiente, raggiungendo una complessità che è circa $\mathcal{O}(k^{1,5})$.

Proposizione 2.11 – Complessità della divisione

Consideriamo l'algoritmo della divisione con resto fra interi a/b, allora

$$T(a/b) \in \mathcal{O}(k^2).$$

Dimostrazione. L'algoritmo della divisione con resto è quello "classico" della divisione in colonna. Ad ogni passaggio si confronta il dividendo con il resto della divisione precedente; questo confronto, nel caso binario, può dare 1 se il divisore coincide con il dividendo o 0 altrimenti. Dopodiché si fanno h moltiplicazioni del risultato del confronto per le h cifre di b e si sottrae il risultato al resto precedente.
Complessivamente si fanno h moltiplicazioni per la lunghezza del quoziente e altrettante sottrazioni. Le sottrazioni sono di un ordine più basso rispetto alle moltiplicazioni, resta quindi da stimare la lunghezza del quoziente.

$$a = b\,q + r \implies L(a) = L(b\,q) + L(r).$$

Osserviamo che, in generale, se $a \geqslant b$, avremo

$$L(a+b) = \begin{cases} L(a) \\ L(a)+1 \end{cases} \qquad \text{e} \qquad L(a\,b) = \begin{cases} L(a)+L(b)-1 \\ L(a)+L(b) \end{cases}$$

Considerando i casi computazionalmente peggiori e ricordando che $r < b$, avremo

$$L(a) = L(b\,q) + L(r) = L(b\,q) + 1 = L(b) + L(q) + 1 \implies L(q) = L(a) - L(b) < L(a).$$

Per cui abbiamo $h\,k$ operazioni elementari, da cui

$$T(a/b) \in \mathcal{O}(k^2).$$

□

Proposizione 2.12 – Complessità dell'algoritmo euclideo

Consideriamo l'algoritmo euclideo per il calcolo del MCD, allora

$$T\big((a,b)\big) \in \mathcal{O}(k^3).$$

Dimostrazione. In generale possiamo supporre $a > b > 0$. Chiamiamo $r_0 = a$ e $r_1 = b$.

Tramite la divisione col resto otteniamo

$$r_0 = q_1 r_1 + r_2,$$
$$r_1 = q_2 r_2 + r_3,$$
$$\dots$$

Iterando questo procedimento, al passo i avremo

$$r_{i-1} = q_i r_i + r_{i+1}.$$

Dopo un certo numero di passi, che siamo sicuri essere finito poiché $r_1 > r_2 > \dots$, arriveremo a

$$r_{m-1} = q_m r_m + 0 \implies (a, b) = r_m.$$

Quindi nell'algoritmo vengono compiute m divisioni. Poiché di queste ultime conosciamo già il peso computazionale, dobbiamo stimare m in termini di k.
Lo scenario computazionalmente peggiore è quello in cui il resto diminuisca sempre di 1. In questo caso m sarebbe confrontabile con $b \approx 2^h$. Se questa fosse la migliore stima, la complessità di (a, b) sarebbe esponenziale.
Cerchiamo quindi una stima migliore per m. Consideriamo i primi tre passi:

$$a = b\,q + r, \qquad b = q'r + r' \qquad r = q''r' + r'',$$

e mostriamo che $2r \leqslant a$. Ci sono due casi:

- Se $q = 1$ avremo $r = a - b$. Inoltre $q = 1$ può avvenire solo se $b > a/2$, da cui
$$r < \frac{a}{2}.$$
- Se $q > 1$ necessariamente $b \leqslant a/2$, da cui
$$r < b \leqslant \frac{a}{2}.$$

Pertanto $2r \leqslant a$. In generale

$$2r_{i+2} \leqslant r_i \implies m \leqslant \lfloor \log_2 a \rfloor + 1,$$

che ci permette di concludere

$$T\big((a, b)\big) \in \mathcal{O}(k^3).$$ □

2.4 PROBLEMI P E NP

Introduciamo brevemente la classe di problemi P e NP per poi valutarne l'applicabilità ad eventuali algoritmi crittografici.

Definizione 2.13 – Problemi P

Definiamo P la classe di problemi per cui esiste un algoritmo che risolve il dato problema in tempo *polinomiale*.

Definizione 2.14 – Problemi NP

Definiamo NP la classe di problemi per cui esiste un algoritmo che li risolve (non necessariamente in tempo polinomiale) e tali che, un'eventuale soluzione può essere verificata in tempo polinomiale.

Osservazione. Chiaramente si ha $P \subseteq NP$. Uno dei problemi centrali della teoria della complessità è proprio stabilire se

$$P \neq NP.$$

Definizione 2.15 – Problemi NP-completi

Un problema π di classe NP si dice NP-completo se, per ogni problema $\pi' \in NP$, esiste un algoritmo che riduce la soluzione di π' ad una di π in tempo polinomiale.

Osservazione. Se esistesse un algoritmo che risolve un problema NP-completo in tempo polinomiale, si avrebbe immediatamente

$$P = NP.$$

I problemi NP-completi sono quindi, di fatto, i problemi computazionalmente più difficili tra quelli di classe NP.

2.5 PROBLEMA DELLO ZAINO

In questo paragrafo studieremo un problema NP-completo e una sua applicazione nella crittografia a chiave pubblica.

Supponiamo di avere uno zaino che possa contenere b unità e una lista di oggetti di volume $a_1, \ldots, a_k$. Il problema dello zaino è riempire completamente lo zaino, formalmente dobbiamo trovare una k-pla

$$e = (e_1, \ldots, e_k) \qquad \text{con } e_i \in \{0, 1\},$$

tale che

$$b = \sum_{i=1}^{k} e_i a_i;$$

oppure mostrare che tale k-pla non esiste.

É facile osservare che si tratta di un problema computazionalmente difficile, dati k oggetti vi sono infatti 2^k possibili combinazioni di tali oggetti. Nonostante ci siano algoritmi più efficienti, non ne esiste nessuno polinomiale. D'altronde la verifica di una possibile soluzione avviene in tempo lineare, pertanto il problema dello zaino è un problema di classe NP. Si dimostra inoltre essere NP-completo.

Affinché vi sia una rilevanza dal punto di vista crittografico, dobbiamo identificare dei casi in cui la risoluzione del problema dello zaino sia computazionalmente facile. Definiamo quindi un particolare tipo di successioni per cui ciò avviene

Definizione 2.16 – Successioni supercrescenti

Una successione $(a_1, \ldots, a_k)$ si dice *supercrescente* se ogni termine è maggiore della somma dei precedenti, ovvero se

$$a_i > \sum_{j=1}^{i-1} a_j.$$

Il motivo per cui una successione supercrescente rende facile il problema dello zaino è che, per verificare se una soluzione esiste, è sufficiente inserire nello zaino gli oggetti più grandi possibili. Infatti se l'oggetto a_i fosse il più grande inseribile nello zaino, anche se scegliessimo di inserire tutti gli oggetti che lo precedono al suo posto, otterremo comunque una grandezza più piccola di a_i. Questo ci garantisce inoltre che la soluzione, se esiste, è unica.

Esempio. Consideriamo la seguente successione supercrescente

$$(1, 2, 4, 8, 16, 32).$$

Cerchiamo di riempire uno zaino con $b = 45$ utilizzando la strategia enunciata prima, anche detta "greedy":

$$45 = 32 + \mathit{13} = 32 + 8 + \mathit{5} = 32 + 8 + 4 + 1.$$

2.6 CRITTOSISTEMA DI MERKLE–HELLMAN

Il crittosistema di Merkle-Hellman si basa sul problema dello zaino e sfrutta un mascheramento delle successioni supercrescenti per costruire la chiave privata.

L'idea di base è la seguente:

- La chiave privata è costituita da una successione $(a_1, \dots, a_k)$.
- Il messaggio, per essere cifrato, deve essere una stringa binaria $(e_1, \dots, e_k)$ di k elementi.
- Il messaggio cifrato è
$$b = \sum_{i=1}^{k} e_i a_i$$
dove sappiamo che $e_i \in \{0, 1\}$.

In questo modo, per decifrare il messaggio bisognerebbe risolvere un problema dello zaino. D'altronde in queste condizioni il messaggio non può essere decifrato neppure da Bob.

Dobbiamo quindi sfruttare le proprietà sulle successioni supercrescenti che abbiamo visto nel precedente paragrafo. Chiaramente la chiave pubblica non può essere una successione supercrescente, altrimenti chiunque potrebbe decifrare il messaggio. É necessario mascherare questa proprietà.

Bob sceglie una successione supercrescente $(a_1, \dots, a_k)$. Sceglie inoltre due interi n e $u < n$ tali che

$$n > \sum_{i=1}^{k} a_i \qquad \text{e} \qquad (u, n) = 1.$$

Attraverso questi ultimi costruisce una nuova successione $(a_1^*, \dots, a_k^*)$ tale che

$$a_i^* \equiv u\, a_i \pmod{n}.$$

Così facendo avremo

- $(a_1, \dots, a_k), n$ e u la chiave privata.
- $(a_1^*, \dots a_k^*)$ la chiave pubblica.

La cifratura avviene come descritto prima, Alice invia

$$b^* = \sum_{i=1}^{k} e_i a_i^*.$$

Per decifrare, Bob calcola v, l'inverso moltiplicativo di u modulo n. Tramite v calcola

$$b = v\, b^* \pmod{n}.$$

Ora

$$b \equiv v\, b^* \equiv v \sum_{i=1}^{k} e_i a_i^* \equiv v \sum_{i=1}^{k} e_i u\, a_i \equiv v\, u \sum_{i=1}^{k} e_i a_i$$
$$\equiv \sum_{i=1}^{k} e_i a_i \pmod{n}$$

Inoltre sia b che $\sum_i e_i a_i$ sono minori di n, per cui

$$b \equiv_n \sum_{i=1}^{k} e_i a_i \implies b = \sum_{i=1}^{k} e_i a_i.$$

A questo punto Bob deve risolvere un problema dello zaino facile per decifrare il messaggio.

Esempio. Come chiave privata utilizziamo

$$(1, 3, 7, 15, 31, 63, 127, 255), \qquad n = 557, \qquad u = 323.$$

Osserviamo che, come da condizione, n è maggiore della somma della successione e u è coprimo con n. Moltiplicando ogni termine della successione supercresente per u e riducendo modulo n, si ottiene la chiave pubblica

$$(323, 412, 33, 389, 544, 297, 360, 486).$$

Supponiamo di voler cifrare la stringa binaria 01100101 che ha lunghezza 8 come la chiave. Otterremo la codifica

$$b^* = 0 \cdot 323 + 1 \cdot 412 + 1 \cdot 33 + 0 \cdot 389 + 0 \cdot 544 + 1 \cdot 297 + 0 \cdot 360 + 1 \cdot 486 = 1228.$$

Bob riceve b^*. Calcola $v = 169$ l'inverso di u modulo n per ottenere

$$b = v\, b^* \equiv 328 \pmod{n}.$$

Risolve il problema dello zaino con $b = 328$ e la sua successione supercrescente:

$$328 = 255 + \mathit{73} = 255 + 63 + \mathit{10} = 255 + 63 + 7 + 3.$$

Confrontando con i termini presenti nella successione supercrescente riottiene la stringa 01100101.

Il crittosistema che abbiamo descritto in questo paragrafo è stato sviluppato da Merkle e Hellman nel 1978. É un sistema molto elegante, molto più semplice di sistemi come l'RSA. D'altronde venne violato piuttosto in fretta, nel 1984 Shamir pubblicò un articolo in cui esponeva un algoritmo che forzava il crittosistema in tempo polinomiale.

3 IL CRITTOSISTEMA RSA

In questo capitolo descriveremo in dettaglio il crittosistema RSA. Nella parte iniziale richiameremo e svilupperemo alcuni concetti matematici e computazionale che ci saranno utili in seguito.

3.1 INTRODUZIONE

In questo paragrafo richiameremo alcuni fatti di teoria modulare e analizzeremo l'algoritmo di esponenziazione modulare.

Teorema 3.1 – di Eulero-Fermat

Siano $a \in \mathbb{Z}$ e $n \in \mathbb{N}$ tali che $(a, n) = 1$. Allora

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

Osservazione. Un caso particolare si ha quando $n = p$ primo. In tal caso se $p \nmid a$ si ha $(a, p) = 1$ e in particolare

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

Notazione. $\varphi(n)$ è la funzione di Eulero calcolata in n, ovvero il numero di interi minori di n e coprimi con esso. Ricordiamo che in generale se $n = p_1^{\alpha_1} \cdot \ldots \cdot p_s^{\alpha_s}$ avremo

$$\varphi(n) = \varphi(p_1^{\alpha_1}) \cdot \ldots \cdot \varphi(p_s^{\alpha_s}) = p_1^{\alpha_1 - 1}(p_1 - 1) \cdot \ldots \cdot p_s^{\alpha_s - 1}(p_s - 1).$$

Per cui è possibile calcolare $\varphi(n)$ se si conosce la fattorizzazione di n.

Corollario. Sia $n = p\,q$ con p, q primi. Sia $m \equiv 1 \pmod{\varphi(n)}$. Allora

$$a^m \equiv a \pmod{n}, \forall\, a \in \mathbb{Z}.$$

Dimostrazione. La tesi è banalmente vera per Eulero se $(a, n) = 1$. Supponiamo quindi che $(a, n) > 1$. Passando in modulo p, vi sono due possibilità:

- Se $p \mid a$ allora
$$a^m \equiv a \equiv 0 \pmod{p}.$$
- Se $p \nmid a$, per ipotesi abbiamo $m \equiv 1 \pmod{\varphi(n)}$ dove $\varphi(n) = (p-1)(q-1)$, quindi
$$a^m = a^{1+k(p-1)(q-1)} = a\,(a^{p-1})^{k(q-1)} \equiv a \pmod{p}.$$

Analogamente si mostra $a^m \equiv a \pmod{q}$, da cui

$$\begin{cases} a^m \equiv a \pmod{p} \\ a^m \equiv a \pmod{q} \end{cases} \implies a^m \equiv a \pmod{n},$$

per il teorema cinese dei resti. □

In generale, se vogliamo calcolare a^n, dobbiamo eseguire $n-1$ moltiplicazioni. Se $n \approx 2^k$ questo significa che la complessità dell'esponenziazione è esponenziale. Per ridurre tale complessità, l'idea è scrivere n come somma di potenze di 2 e spezzare la potenza. Supponiamo di avere

$$n = 2^k + 2^{k-i} + \ldots + 2^{k-m} \implies a^k = a^{2^k} a^{2^{k-i}} \cdot \ldots \cdot a^{2^{k-m}}.$$

A questo punto è sufficiente calcolare le potenze di a fino a a^{2^k}, nel processo calcoleremo incidentalmente anche tutte quelle precedenti. Infine basta moltiplicare fra di loro le potenze trovate. Osserviamo in particolare che per calcolare ogni potenza è sufficiente una singola moltiplicazione, infatti

$$\begin{aligned} a^2 &= a \cdot a; \\ a^{2^2} &= a^2 \cdot a^2; \\ &\ldots \\ a^{2^k} &= a^{2^{k-1}} \cdot a^{2^{k-1}}. \end{aligned}$$

A noi interessa valutare la complessità di questo algoritmo nel caso modulare

Proposizione 3.2 – Complessità dell'algoritmo square and multiply

Siano $b, m, n \in \mathbb{Z}$ con $L(m) = k$ e $L(n) = h$. Consideriamo l'algoritmo A per il calcolo di $b^m \pmod n$ tramite la strategia precedente, allora

$$T(A) \in \mathcal{O}\big((k-1)h^2\big).$$

Dimostrazione. Supponiamo di avere

$$m = a_0 + a_1 2 + a_2 2^2 + \ldots + a_{k-1} 2^{k-1} \qquad \text{con } a_i \in \{0, 1\}.$$

Pertanto avremo

$$b^m = b^{a_0 + a_1 2 + \ldots + a_{k-1} 2^{k-1}} = b^{a_0} b^{a_1 2} \cdot \ldots \cdot b^{a_{k-1} 2^{k-1}}.$$

Dobbiamo calcolare le k potenze di 2. Come osservato in precedenza il calcolo di 2^i è costituito da una singola moltiplicazione di numeri minori di n. Nel caso computazionalmente peggiore, ovvero se $a_i = 1$ per ogni i, avremo $k-1$ passi. Quindi, ricordando la complessità della moltiplicazione, avremo

$$T(A) \in \mathcal{O}\big((k-1)h^2\big).$$

□

Osservazione. Se si ha $(b, n) = 1$, possiamo supporre che $m \leqslant \varphi(n)$. Infatti se così non fosse, potrei scrivere

$$b^m \equiv b^{m'} \pmod n \qquad \text{con } m \equiv m' \pmod{\varphi(n)}.$$

Pertanto se $(b, n) = 1$ avremo $m \leqslant \varphi(n) \leqslant n$, ovvero $L(m) \leqslant L(n)$. Quindi la complessità dell'esponenziazione può essere stimata completamente in termini di $h = L(n)$, ottenendo

$$T(A) \in \mathcal{O}(h^3).$$

3.2 DESCRIZIONE DEL CRITTOSISTEMA

Come ogni crittosistema a chiave pubblica, anche RSA si basa su una funzione unidirezionale speciale. L'idea del crittosistema si basa sulla difficoltà nella fattorizzazione dei numeri.

In particolare abbiamo visto come moltiplicare due interi a n bit sia dell'ordine di $\mathcal{O}(n^2)$, mentre è possibile dimostrare che fattorizzare un numero a n bit è un'operazione dell'ordine di $\mathcal{O}(2^{c\, n^{1/3}})$.

Il crittosistema RSA si schematizza come segue:

- Sia $N = p\, q$ con p, q primi e siano $P = C = \mathbb{Z}_N$.
- Lo spazio delle chiavi è

$$K = \{\, (N, p, q, d, e) \mid d\, e \equiv 1 \pmod{\varphi(N)} \,\}.$$

Da questo insieme avremo

- (p, q, d) la chiave privata.
- (N, e) la chiave pubblica.

Se $k = (N, p, q, d, e)$ è una chiave, il messaggio $x \in P$ verrà criptato attraverso

$$e_k(x) = x^e \pmod{N}.$$

Viceversa, un codice $y \in C$ sarà decriptato con

$$d_k(y) = y^d \pmod{N}.$$

Dalla costruzione della chiave, sappiamo che $e\, d \equiv 1 \pmod{\varphi(N)}$, inoltre $N = p\, q$ con p, q primi. Quindi per il corollario di Fermat sappiamo che

$$(x^e)^d \equiv x \pmod{N} \iff d_k(e_k(x)) = x.$$

Esempio. Supponiamo che Bob scelga $p = 17$ e $q = 11$, costruisce quindi

$$N = 17 \cdot 11 = 187 \qquad \text{e} \qquad \varphi(N) = 16 \cdot 10 = 160.$$

Bob scegli inoltre $e = 7$ e calcola $d = e^{-1} = 23 \pmod{160}$. Pubblica quindi la chiave pubblica $(187, 7)$.
Supponiamo che Alice voglia cifrare il testo in chiaro 88, il cifrato sarà quindi

$$88^7 = 11 \pmod{187}.$$

Bob riceve 11 da Alice, per decifrare calcola

$$11^{23} = 88 \pmod{187}$$

ritrovando quindi il messaggio originale.

Alla base dell'RSA vi sono quindi una serie di problemi "facili" in fase di cifratura e "difficili" in decifratura. Nei prossimi paragrafi ci occuperemo proprio di studiare algoritmi polinomiali per i problemi di cifratura e di dimostrare che tutti i problemi di decifratura sono equivalenti alla fattorizzazione di un intero.

In particolare i problemi che coinvolgono la cifratura sono

- Determinare se un intero n è primo.
- Dati a e n, trovare (e, n) e, nel caso $(e, n) = 1$, calcolare l'inverso di e modulo n.
- Calcolare la funzione $x^e \pmod{n}$.

Degli ultimi due problemi abbiamo già esibito un algoritmo polinomiale, rispettivamente l'algoritmo di Euclide e l'algoritmo square and multiply. Ai cosiddetti test di primalità, dedicheremo il prossimo paragrafo.

Viceversa, i problemi che coinvolgono la decifratura sono

- Fattorizzare un intero n.
- Dato un intero n, calcolare $\varphi(n)$.
- Dati n, e, trovare d tale che
$$(x^e)^d \equiv x \pmod{n}.$$

A prima vista questi problemi sono esposti in ordine di difficoltà. É chiaro infatti che se si riesce a fattorizzare n, gli altri due problemi diventano banali. Questo ci fa capire che violare l'RSA non può essere più difficile che fattorizzare un intero. D'altronde, vedremo in seguito che questi tre problemi sono tra loro equivalenti.

3.3 TEST DI PRIMALITÀ

In questo paragrafo ci occuperemo di descrivere alcuni algoritmi polinomiali per determinare se un interno n sia primo. Come è facile intuire, affinché questi algoritmi siano polinomiali, essi non forniranno alcuna informazione sulla fattorizzazione di n.

Nella prima parte faremo cenno ad alcuni elementi di teoria dei numeri, per una trattazione più approfondita si consiglia di fare riferimento ad un testo specializzato.

Teorema 3.3 – Cardinalità numeri primi

Vi sono infiniti numeri primi.

Dimostrazione. Supponiamo per assurdo che vi siano un numero finito di primi $p_1, \ldots, p_k$. Definiamo
$$N = p_1 \cdot \ldots \cdot p_k + 1.$$
Per costruzione N non è divisibile per nessuno dei $p_1, \ldots, p_k$. Per cui vi sono altri primi che fattorizzano N, da cui l'assurdo. □

Teorema 3.4 – dei numeri primi

Sia $\pi(x)$ la funzione aritmetica che enumera i numeri primi fino ad x. Allora
$$\lim_{x\to\infty} \frac{\pi(x)}{\frac{x}{\ln x}} = 1$$

Osservazione. Tramite il teorema dei numeri primi, è possibile stimare la probabilità che un numero N sia primo. Tale probabilità è infatti circa
$$\pi(N)/N.$$
Ad esempio, la probabilità che un numero casuale con al più cento cifre decimali sia primo è circa
$$\frac{1}{\ln(10^{100})} = \frac{1}{230}.$$
Questa probabilità può essere raffinata, scartando ad esempio i numeri pari e i multipli di tre, scarteremmo $\frac{2}{3} = \frac{1}{2} + \frac{1}{6}$. La probabilità di trovare un primo con al più cento cifre diventa quindi $\frac{1}{77}$.
Questo ci dice che trovare numeri primi di grandi dimensioni è relativamente facile.

Definizione 3.5 – Numero di Fermat

Un numero si dice *di Fermat* se è della forma

$$F_n = 2^{2^n} + 1.$$

Esempio. $F_0 = 3, F_1 = 5, F_2 = 17$ sono numeri di Fermat.

Osservazione. Fermat congetturò che F_n è primo per ogni n. Ciò venne velocemente dimostrato falso fattorizzando F_5.

Proposizione 3.6 – Espressione ricorsiva dei numeri di Fermat

Siano F_i i numeri di Fermat, allora

$$F_n - 2 = \prod_{i=0}^{n-1} F_i.$$

Dimostrazione. Mostriamolo per induzione:

- Per $n = 1$ è banalmente vero, infatti
$$F_1 - 2 = 5 - 2 = 3 = F_0.$$
- Supponiamo che la tesi sia vera per $n-1$ e mostriamola per n:
$$\begin{aligned} F_n - 2 &= 2^{2^n} + 1 - 2 = 2^{2^n} - 1 = \left(2^{2^{n-1}}\right)^2 - 1 = \left(2^{2^{n-1}} - 1\right)\left(2^{2^{n-1}} + 1\right) \\ &= \left(2^{2^{n-1}} + 1 - 2\right)\left(2^{2^{n-1}} + 1\right) = (F_{n-1} - 2)\, F_{n-1}. \end{aligned}$$
Ma per ipotesi induttiva
$$F_{n-1} - 2 = \prod_{i=0}^{n-2} F_i,$$
da cui
$$F_n - 2 = \left(\prod_{i=0}^{n-2} F_i\right) F_{n-1} = \prod_{i=0}^{n-1} F_i.$$ □

Corollario. Siano F_n, F_m due numeri di Fermat con $n \neq m$. Allora

$$(F_n, F_m) = 1.$$

Dimostrazione. Supponiamo per assurdo che p sia un primo che divide sia F_n che F_m, dove assumiamo $n < m$ senza perdita di generalità. Dal momento che $p \mid F_n$ avremo

$$p \mid F_0 \cdot \ldots \cdot F_n \cdot \ldots \cdot F_{m-1}.$$

Inoltre, dalla proposizione precedente,

$$p \mid F_m = F_0 \cdot \ldots \cdot F_{m-1} + 2.$$

Quindi p deve dividere la differenza fra $F_0 \cdot \ldots \cdot F_{m-1} + 2$ e $F_0 \cdot \ldots \cdot F_{m-1}$ che è 2. Da cui $p = 2$ che è assurdo in quanto i numeri di Fermat sono per costruzione dispari. . □

Osservazione. Da questo corollario segue una dimostrazione alternativa sull'infinità dei numeri primi: infatti, dal momento che esistono infiniti numeri di Fermat e che ogni numero primo ne divide al più uno solo, devono esistere infiniti numeri primi.

Definizione 3.7 – Test deterministico

Un test si dice *deterministico* se risponde in modo univoco ad un problema, senza margine di errore.

Definizione 3.8 – Test probabilistico

Un test si dice *probabilistico* se consiste di una successione di test $\{T_m\}_{m\in\mathbb{N}}$ e una successione che tende a zero $\{\varepsilon_m\}_{m\in\mathbb{N}}$ tale che:

- Se il problema non passa un test T_m allora la risposta è certamente negativa.
- La probabilità che il problema superi di test $T_1, \dots, T_m$ ma che la risposta sia negativa è minore di ε_m.

Osservazione. Fino al 2002 gli unici test di primalità polinomiali erano probabilistici. Nel 2002, venne trovato un algoritmo polinomiale per determinare la primalità con una complessità computazionale di $\mathcal{O}(n^{12})$ in seguito migliorata a $\mathcal{O}(n^6)$. Da ciò sappiamo che il problema di primalità è un problema P.

Dal punto di vista dell'efficienza si preferisce usare un test di primalità probabilistico, in quanto più veloce e sufficientemente affidabile. L'idea di un test di primalità probabilistico è quello di cercare prove del fatto che n si comporti o meno come un primo senza cercare i suoi fattori. Si utilizzano quindi condizioni necessarie dei numeri primi, che in caso di fallimento ci garantiscono la non primalità di n, mentre in caso di successo aumentano la probabilità che n sia primo.

3.3.1 TEST DI FERMAT

Il test di Fermat è un test di primalità probabilistico che sfrutta il teorema di Fermat. Ricordiamo che quest'ultimo afferma che, se p è primo e $1 \leqslant a < p$, allora

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

Sia n è l'intero di cui vogliamo determinare la primalità, se troviamo $a < n$ tale che

$$a^{n-1} \not\equiv 1 \pmod{n},$$

allora sappiamo per certo che n non è primo pur non avendo studiato i suoi fattori.

Il test di Fermat per $n \in \mathbb{N}$ è quindi il seguente: prendiamo $a < n$ e calcoliamo (a, n),

- Se $(a, n) \neq 1$ allora n non è primo;
- Se $(a, n) = 1$ calcoliamo a^{n-1} modulo n. Se è diverso da 1 allora n non è primo.

Esempio. Valutiamo la primalità di 323:

$$2^{322} \equiv 157 \pmod{323},$$

quindi 323 non è primo.

Definizione 3.9 – Pseudoprimo

Si dice che n è uno *pseudoprimo in base* a, se n non è primo ma

$$a^n \equiv a \pmod{n}.$$

Osservazione. Il nome pseudoprimo si usa in quanto il teorema di Fermat non fornisce una condizione sufficiente di primalità.

Esempio. 341 è uno pseudoprimo in base 2, infatti

$$2^{340} \equiv 1 \pmod{341},$$

ma $341 = 11 \cdot 31$.

Definizione 3.10 – Numero di Carmichael

Un intero n non primo si dice *numero di Carmichael* se è uno pseudoprimo in base a per ogni

$$1 < a < n \qquad \text{tale che } (a, n) = 1.$$

Esempio. 561 è il più piccolo numero di Carmichael.

Osservazione. L'esistenza di tali numeri ci dice che il test di Fermat non è affidabile come test di primalità. Ad ogni modo esso fornisce l'idea generale dei test di primalità probabilistici.

Teorema 3.11 – Cardinalità dei numeri di Carmichael

Ci sono infiniti numeri di Carmichael.

Dimostrazione. Non fornita. □

Proprietà 3.12. Sia n un numero di Carmichael. Allora n è prodotto di primi distinti, ovvero

$$n = p_1 p_2 \cdot \ldots \cdot p_s \qquad \text{con } p_i \neq p_j \; \forall \, i \neq j.$$

Proprietà 3.13. Sia $n = p_1 \cdot \ldots \cdot p_s$ prodotto di primi distinti. Allora n è un numero di Carmichael se e soltanto se

$$p - 1 \mid n - 1 \; \forall \; p \mid n.$$

3.3.2 TEST DI SOLOVAY–STRASSEN

Questo test di primalità, nonostante sia stato superato da test più recenti, ha una grande importanza storica in quanto dimostrò la possibilità di utilizzo del crittosistema RSA.

Per descriverlo richiamiamo di seguito alcuni concetti di teoria dei numeri, in particolare i residui quadratici, il simbolo di Legendre e quello di Jacobi.

Definizione 3.14 – Residuo quadratico

Sia p un primo dispari. $a \in \mathbb{Z}$ si dice *residuo quadratico modulo* p se la congruenza

$$x^2 \equiv a \pmod{p}$$

ha soluzione.

Esempio. Gli interi $1, 3, 4, 5, 9$ sono residui quadratici modulo 11.

Osservazione. Modulo p, i residui quadratici sono in numero $\frac{p-1}{2}$. Precisamente sono

$$1^2, 2^2, \ldots, \left(\frac{p-1}{2}\right)^2.$$

Questa osservazione segue da

$$a^2 \equiv b^2 \iff a^2 - b^2 \equiv 0 \iff (a-b)(a+b) \equiv 0 \pmod{p},$$

da cui

$$a + b \equiv 0 \pmod{p} \qquad \text{oppure} \qquad a - b \equiv 0 \pmod{p},$$

che ci dice $a \equiv \pm b \pmod{p}$.

Notazione. Con $\mathbb{Z}_p^* := U(\mathbb{Z}_p)$ indichiamo il gruppo moltiplicativo di $\mathbb{Z}_p$.

Osservazione. $\mathbb{Z}_p$ è un campo, pertanto $\mathbb{Z}_p^*$ è un gruppo moltiplicativo ciclico.

Definizione 3.15 – Radice primitiva

Consideriamo il gruppo moltiplicativo ciclico $\mathbb{Z}_p^*$. Un generatore g di $\mathbb{Z}_p^*$ si dice *radice primitiva modulo* p.

Proprietà 3.16. Sia g una radice primitiva modulo p. Allora i residui quadratici modulo p sono della forma g^k con k pari.

Proposizione 3.17 – Criterio di Eulero

Sia p un primo dispari e sia $a \in \mathbb{Z}$ tale che $p \nmid a$. Allora a è un residuo quadratico modulo p se e soltanto se

$$a^{\frac{p-1}{2}} \equiv 1 \pmod{p}.$$

Dimostrazione. Dal teorema di Fermat, sappiamo che

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

Da ciò segue

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

Sia g una radice primitiva modulo p, pertanto $a = g^k$ per qualche k. Segue

$$a^{\frac{p-1}{2}} = g^{k\frac{p-1}{2}} \equiv 1 \pmod{p} \iff p-1 \mid k\frac{p-1}{2} \iff k \text{ pari.}$$

$p-1$ *è l'ordine di* g

Come abbiamo visto nella proprietà precedente, questa è una condizione necessaria e sufficiente affinché a sia un residuo quadratico modulo p. □

Osservazione. Viceversa a non è un residuo quadratico modulo p se e soltanto se

$$a^{\frac{p-1}{2}} \equiv -1 \pmod{p}.$$

Si mostra in maniera analoga.

Definizione 3.18 – Simbolo di Legendre

Sia p un primo dispari e sia $a \in \mathbb{Z}$. Definiamo il *simbolo di Legendre* come segue

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

Proprietà 3.19.

$$a \equiv b \pmod{p} \implies \left(\frac{a}{p}\right)_L = \left(\frac{b}{p}\right)_L.$$

Proprietà 3.20.

$$p \nmid a \implies \left(\frac{a^2}{p}\right)_L = 1.$$

Proprietà 3.21.

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

Proprietà 3.22.

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

Proprietà 3.23.

$$\left(\frac{1}{p}\right)_L = 1.$$

Proprietà 3.24.

$$\left(\frac{-1}{p}\right)_L = (-1)^{\frac{p-1}{2}} = \begin{cases} 1 & p \equiv 1 \pmod 4 \\ -1 & p \equiv 3 \pmod 4 \end{cases}$$

Proprietà 3.25.

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

Teorema 3.26 – Legge di reciprocità quadratica

Siano p e q primi dispari distinti. Allora

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

Osservazione. Il teorema può essere letto anche nel modo seguente:

- Se p o q sono congrui a 1 modulo 4, allora p è un residuo quadratico modulo q se e soltanto se q è un residuo quadratico modulo p.
- Se p e q sono congrui a 3 modulo 4, allora p è un residuo quadratico modulo q se e soltanto se q non è un residuo quadratico modulo p.

Di seguito descriveremo il simbolo di Jacobi, questo estende il simbolo di Legendre ad n qualsiasi non necessariamente primo.

Definizione 3.27 – Simbolo di Jacobi

Sia $n = p_1^{\alpha_1} \cdot \ldots \cdot p_s^{\alpha_s}$ dispari e sia $a \in \mathbb{Z}$. Il *simbolo di Jacobi* di è definito come

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)_L^{\alpha_1} \cdot \ldots \cdot \left(\frac{a}{p_s}\right)_L^{\alpha_s}.$$

Osservazione. Il simbolo di Jacobi nasce con l'intento di semplificare il calcolo del simbolo di Legendre. D'altronde non ci fornisce informazioni sul fatto che a sia o meno un residuo quadratico modulo n.

Esempio. Calcoliamo il simbolo di Jacobi 2 su 15:

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

ma 2 non è un quadrato modulo 15.

Si dimostra che molte delle proprietà del simbolo di Legendre valgono ugualmente per il simbolo di Jacobi:

Proprietà 3.28.

$$a \equiv b \pmod n \implies \left(\frac{a}{n}\right) = \left(\frac{b}{n}\right).$$

Proprietà 3.29.

$$(a, n) = 1 \implies \left(\frac{a^2}{n}\right) = 1.$$

Proprietà 3.30.

$$\left(\frac{a\,b}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right).$$

Proprietà 3.31.

$$\left(\frac{1}{n}\right) = 1.$$

Proprietà 3.32.

$$\left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}} = \begin{cases} 1 & n \equiv 1 \pmod 4 \\ -1 & n \equiv 3 \pmod 4 \end{cases}$$

Proprietà 3.33.

$$\left(\frac{2}{n}\right)_L = (-1)^{\frac{n^2-1}{8}} = \begin{cases} 1 & n \equiv \pm 1 \pmod 8 \\ -1 & n \equiv \pm 3 \pmod 8 \end{cases}$$

Proprietà 3.34. Se $m, n \in \mathbb{Z}$ sono dispari, allora

$$\left(\frac{n}{m}\right)\left(\frac{m}{n}\right) = (-1)^{\frac{n-1}{2}\frac{m-1}{2}}.$$

Proposizione 3.35 – Complessità computazionale del simbolo di Jacobi

Consideriamo l'algoritmo A per il calcolo del simbolo di Jacobi di a su n, allora

$$T(A) \in \mathcal{O}(k^3).$$

Dimostrazione. Sia k il massimo tra $L(a)$ e $L(n)$. Ad ogni passo del calcolo di $\left(\frac{a}{n}\right)$ dovremo

1. Ridurre il numeratore modulo il denominatore: complessità $\mathcal{O}(k^2)$.
2. Se il numeratore è pari, bisogna estrarre la parte pari e valutarla. La valutazione è una divisione per 8: complessità $\mathcal{O}(k)$.
3. Scambiare il numeratore col denominatore tramite la legge di reciprocità e successivamente valutare il segno. La valutazione è una divisione per 4: complessità $\mathcal{O}(k)$.

L'algoritmo termina in al più k passaggi, pertanto la complessità totale è

$$\mathcal{O}(k^3).$$

□

Ricordiamo, dalle proprietà sul simbolo di Legendre che, se p è un primo dispari, allora

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

Sia n un intero di cui vogliamo determinare la primalità, se troviamo $a < n$ tale che

$$\left(\frac{a}{n}\right) \not\equiv a^{\frac{n-1}{2}} \pmod{n},$$

allora n non è primo.

Osservazione. Chiaramente questa non è una condizione sufficiente. Ad esempio 25 non è primo, ma

$$\left(\frac{7}{25}\right) = \left(\frac{25}{7}\right) = \left(\frac{4}{7}\right) = \left(\frac{2^2}{7}\right) = 1.$$

Il test di Solovay-Strassen per $n \in \mathbb{N}$ dispari è il seguente: prendiamo $a < n$ e calcoliamo (a, n),

- Se $(a, n) \neq 1$ allora n non è primo;
- Se $(a, n) = 1$ calcoliamo $\left(\frac{a}{n}\right)$ e $a^{\frac{n-1}{2}}$ modulo n. Se non coincidono allora n non è primo.

Definizione 3.36 – Pseudoprimo di Eulero

Un intero n si dice *pseudoprimo di Eulero in base* a, se n non è primo, $(a, n) = 1$ e

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

Notazione. Per dire che n è uno pseudoprimo di Eulero in base a scriveremo

$$n \in \mathsf{PSPE}(a).$$

Per gli pseudoprimi semplici scriviamo $n \in \mathsf{PSP}(a)$.

Osservazione. Ogni n dispari composito è uno pseudoprimo di Euelero in base ± 1.

Proposizione 3.37 – Pseudprimo di Eulero è pseudoprimo

Sia n uno pseudoprimo di Eulero in base b, allora n è uno pseudoprimo in base b.

Dimostrazione. Per definizione

$$n \in \mathsf{PSPE}(b) \iff \left(\frac{b}{n}\right) \equiv b^{\frac{n-1}{2}} \pmod{n}.$$

Ora $(b, n) = 1$, quindi

$$\left(\frac{b}{n}\right) = \pm 1 \implies b^{\frac{n-1}{2}} \equiv \pm 1 \pmod{n},$$

ne segue

$$1 \equiv \left(b^{\frac{n-1}{2}}\right)^2 \equiv b^{n-1} \pmod{n},$$

ovvero $n \in \mathrm{PSP}(a)$ □

Osservazione. Il viceversa è falso. Infatti

$$91 \in \mathrm{PSP}(3) \qquad \text{ma} \qquad 91 \notin \mathrm{PSPE}(3).$$

Lemma 3.38. Sia n dispari non quadrato perfetto. Allora esiste $b < n$ con $(b, n) = 1$ tale che

$$\left(\frac{b}{n}\right) = -1.$$

Dimostrazione. Supponiamo che n sia primo. La tesi in questo caso è banale poiché vi sono $\frac{n-1}{2}$ non residui quadratici modulo n, per ognuno dei quali si ha

$$\left(\frac{b}{n}\right) = -1.$$

Supponiamo quindi che n sia composito. Per ipotesi n non è un quadrato perfetto, pertanto $n = p^e s$ con p primo, e dispari e $(p, s) = 1$. Sia t un non residuo quadratico modulo p. Consideriamo il seguente sistema di congruenze:

$$\begin{cases} x \equiv t \pmod{p} \\ x \equiv 1 \pmod{s} \end{cases}$$

Dal momento che $(p, s) = 1$, tale sistema ha soluzione unica modulo $p\,s$ per il teorema cinese dei resti. Chiamiamo b tale soluzione. Mostriamo che b soddisfa la tesi:

$b < n$ *in quanto* b *è modulo* $p\,s$

$$\left(\frac{b}{n}\right) = \left(\frac{b}{p}\right)^e \left(\frac{b}{s}\right) = \left(\frac{t}{p}\right)^e \left(\frac{1}{s}\right) = \left(\frac{t}{p}\right)^e = (-1)^e = -1$$

in quanto e è dispari. □

Proposizione 3.39 – Non esistenza di pseudoprimi di Eulero per qualsiasi base

Sia n dispari composito. Allora esiste $1 < b < n$ con $(b, n) = 1$ tale che n non è uno pseudoprimo di Eulero in base b.

Dimostrazione. Supponiamo per assurdo che per ogni b, con $(b, n) =$ e $b < n$, si abbia che n è uno pseudoprimo di Eulero in base b. Per la proposizione precedente ciò implica che n è uno pseudoprimo in base b. Ma se ciò accade per ogni possibile base b, allora n è un numero di Carmichael. Sappiamo, da una precedente proprietà, che tali numeri sono prodotti di primi distinti, pertanto

$$n = p_1 \cdot \ldots \cdot p_l.$$

Per il lemma precedente esisterà $\overline{b}$ tale che

$$\left(\frac{\overline{b}}{n}\right) = -1.$$

Per tale base $\overline{b}$ avremo, in particolare

$$\left(\frac{\overline{b}}{n}\right) \equiv \overline{b}^{\frac{n-1}{2}} \equiv -1 \pmod{n}.$$

Consideriamo il seguente sistema di congruenze:

$$\begin{cases} x \equiv \overline{b} \pmod{p_1} \\ x \equiv 1 \pmod{p_2 \cdot \ldots \cdot p_l} \end{cases}$$

Dal momento che $(p_1, p_2 \cdot \ldots \cdot p_l) = 1$, per il teorema cinese dei resti esiste una soluzione a modulo $p_1 \cdot \ldots \cdot p_l = n$. Mostriamo che n non è uno pseudoprimo di Eulero modulo n: per costruzione

$$a^{\frac{n-1}{2}} \equiv \overline{b}^{\frac{n-1}{2}} \equiv -1 \pmod{p_1},$$

mentre

$$a^{\frac{n-1}{2}} \equiv 1 \pmod{p_2 \cdot \ldots \cdot p_l}.$$

Ne segue che

$$a^{\frac{n-1}{2}} \not\equiv \pm 1 \pmod{n}$$

poiché altrimenti verrebbe violata una delle due congruenze precedenti. Quindi

$$\left(\frac{a}{n}\right) \not\equiv a^{\frac{n-1}{2}} \pmod{n},$$

ovvero n non è uno pseudoprimo di Eulero in base a. Ciò è assurdo, da cui la tesi. □

Osservazione. Questa proposizione ci garantisce che non esistono numeri analoghi a quelli di Carmichael per gli psuedoprimi di Eulero.

Lemma 3.40. Sia n dispari composito e siano b_1, b_2, con $1 < b_1, b_2 < n$ e $(b_1, n) = (b_2, n) = 1$, tali che n è uno pseudoprimo di Eulero in base b_1 e b_2, allora

$$n \in \mathrm{PSPE}(b_1 b_2) \qquad \text{e} \qquad n \in \mathrm{PSPE}(b_1 b_2^{-1}),$$

con b_2^{-1} l'inverso di b_2 modulo n.

Proposizione 3.41 – Stima del numero di basi psudoprime di Eulero per un intero

Sia n dispari composito. Allora la basi b, con $(b, n) = 1$ e $b < n$, tali che n è uno pseudoprimo di Eulero in base b, sono non più della metà di tutte le possibili basi b.

Dimostrazione. Sia $a < n$ con $(a, n) = 1$ tale che n non è uno pseudoprimo di Eulero modulo a. Sia inoltre P definito come l'insieme

$$P = \{\, b \in U(\mathbb{Z}_n) \mid n \in \mathrm{PSPE}(b) \,\} \subseteq U(\mathbb{Z}_n).$$

Se $b \in P$, allora $a\,b \notin P$, poiché altrimenti si avrebbe $a \in P$ per il lemma precedente. Consideriamo quindi la seguente applicazione

$$f_a \colon P \longrightarrow U(\mathbb{Z}_p) \setminus P, b \longmapsto a\,b.$$

f_a è banalmente iniettiva, pertanto

$$|P| \leqslant |U(\mathbb{Z}_p) \setminus P|,$$

da cui

$$|P| \leqslant |U(\mathbb{Z}_p) \setminus P| \leqslant |U(\mathbb{Z}_p)| - |P| \implies |P| \leqslant \frac{1}{2}|U(\mathbb{Z}_p)|.$$

□

Osservazione. Questo ci dice che la probabilità che n superi un test di Solovay-Strassen senza essere effettivamente primo è minore di $1/2$.
Dopo n iterazioni positive, la probabilità che il numero testato non sia primo è minore di $1/2^n$.

3.3.3 TEST DI MILLER–RABIN

Sia p un primo dispari e sia $a < p$. Per il piccolo teorema di Fermat sappiamo che

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

Pertanto

$$a^{\frac{p-1}{2}} \equiv \pm 1 \pmod{p}.$$

Se

$$\frac{p-1}{2} \text{ è pari} \qquad \text{e} \qquad a^{\frac{p-1}{2}} \equiv 1 \pmod{p},$$

allora siamo nella stessa situazione precedente e possiamo iterare il procedimento. In generale, fintanto che

$$\frac{p-1}{2^k} \text{ è pari} \qquad \text{e} \qquad a^{\frac{p-1}{2^k}} \equiv 1 \pmod{p}$$

è possibile continuare ad estrarre una radice.

Questo ci dice che se scriviamo $p - 1 = 2^s t$ con t dispari e preso $a < p$ calcoliamo

$$a^t, a^{2t}, a^{2^2 t}, \ldots, a^{2^{s-1} t}, a^{2^s t} \qquad \text{modulo } s$$

certamente $a^{2^s t} \equiv 1 \pmod{p}$, ma per quanto appena osservato, possiamo trovare un minimo k, con $0 \leqslant k < s$, tale che

$$a^{2^k t} \equiv -1 \pmod{p}.$$

Inoltre, per tutti le esponenziazioni successive a k la congruenza sarà sempre uguale ad 1.

Esempio. Consideriamo $p = 13$, pertanto $p - 1 = 12 = 2^2 \cdot 3$. Prendiamo $a = 2$ e procediamo a calcolare gli esponenziali modulari:

$$2^3 \equiv 8 \not\equiv -1 \pmod{13}$$
$$2^6 \equiv 64 \equiv -1 \pmod{13}.$$

Quindi $k = 1$ e

$$2^{2^h 3} \equiv 1 \pmod{13} \, \forall \, h > 1.$$

Definizione 3.42 – Pseudoprimo forte

Sia n dispari composito. n si dice *pseudoprimo forte in base* b, con $1 < b < n$ e $(b, n) = 1$, se, scritto $n - 1 = 2^s t$ con t dispari, si ha

$$b^t \equiv 1 \pmod{n} \qquad \text{oppure} \qquad \exists \, 0 \leqslant r < s : b^{2^r t} \equiv -1 \pmod{n}.$$

Notazione. Se n è uno pseudoprimo forte in base b, scriviamo

$$n \in \mathrm{PSPF}(b).$$

Esempio. Mostriamo che $25 \in \mathrm{PSPF}(7)$. Iniziamo con lo scrivere $25 - 1 = 2^3 \cdot 3$ e

procediamo calcolando le esponenziazioni modulari di 7:

$$7^3 \equiv 7^2 \cdot 7 \equiv -7 \pmod{25}$$
$$7^6 \equiv (-7)^2 \equiv 49 \equiv -1 \pmod{25}.$$

Proposizione 3.43 – Pseudoprimo forte è pseudoprimo

Sia n dispari composito e sia $1 < b < n$ con $(b, n) = 1$. Allora

$$n \in \mathrm{PSPF}(b) \implies n \in \mathrm{PSP}(b).$$

Dimostrazione. Discende direttamente dalla definizione di pseudoprimo forte. □

Proposizione 3.44 – Relazione tra pseudoprimi forti e di Eulero

Sia n dispari composito e sia $1 < b < n$ con $(b, n) = 1$.

- Se $n \equiv 3 \pmod 4$ allora

$$n \in \mathrm{PSPF}(b) \iff n \in \mathrm{PSPE}(b).$$

- Se $n \equiv 1 \pmod 4$ allora

$$n \in \mathrm{PSPF}(b) \implies n \in \mathrm{PSPE}(b).$$

Dimostrazione. Mostriamo solo il primo punto:

⇐) Se $n \equiv 3 \pmod 4$ si ha necessariamente $n - 1 = 2t$. Pertanto la definizione di pseudoprimo forte in base b diventa

$$b^t \equiv 1 \pmod n \qquad \text{oppure} \qquad b^t \equiv -1 \pmod n,$$

ovvero

$$b^{\frac{n-1}{2}} \equiv \pm 1 \pmod n$$

Inoltre da $n \equiv 3 \pmod 4$ segue

$$\left(\frac{1}{n}\right) = 1 \qquad \text{e} \qquad \left(\frac{-1}{n}\right) = -1.$$

Quindi

$$n \in \mathrm{PSPE}(b) \implies \pm 1 = \left(\frac{b}{n}\right) \equiv b^{\frac{n-1}{2}} \equiv \pm 1 \pmod n,$$

che abbiamo visto essere, in questo particolare caso, ad $n \in \mathrm{PSPF}(b)$.

⇒) Supponiamo $n \in \mathrm{PSPF}(b)$. Osserviamo che

$$n \equiv 3 \pmod 4 \implies \frac{n-3}{4} \in \mathbb{Z} \implies \left(\frac{b^{2\frac{n-3}{4}}}{n}\right) = 1.$$

Pertanto

$$\left(\frac{b}{n}\right) = \left(\frac{b}{n}\right)\left(\frac{b^{2\frac{n-3}{4}}}{n}\right) = \left(\frac{b^{\frac{n-3}{2}+1}}{n}\right) = \left(\frac{b^{\frac{n-1}{2}}}{n}\right).$$

Ora $n \in \mathrm{PSPF}(b)$ e $n \equiv 3 \pmod 4$ ci dicono $b^{\frac{n-1}{2}} \equiv \pm 1 \pmod n$ per quanto visto in precedenza. Per cui

$$\left(\frac{b^{\frac{n-1}{2}}}{n}\right) = \left(\frac{\pm 1}{n}\right) = \pm 1 \equiv b^{\frac{n-1}{2}} \pmod n,$$

quindi

$$\left(\frac{b}{n}\right) \equiv b^{\frac{n-1}{2}} \pmod{n} \implies n \in \mathrm{PSPE}(b). \qquad \square$$

Proposizione 3.45 – Stima del numero di basi pseudoprime forti per un intero

Sia n dispari composito. Allora le basi b, con $(b, n) = 1$ e $b < n$, tali che n è uno pseudoprimo forte in base b, sono non più di un quarto di tutte le possibili basi b.

Dimostrazione. Non fornita. $\square$

Descriviamo di seguito il generico test di Miller-Rabin su un intero n dispari:

1. Scriviamo $n - 1 = 2^s t$ con t dispari.
2. Scegliamo $b < n$ e verifichiamo $(b, n) > 1$, altrimenti n non è primo.
3. Calcoliamo b^t
   - Se $b^t \equiv \pm 1 \pmod{n}$ allora n è un possibile primo.
   - Altrimenti calcolo b^{2t}.
4. Calcoliamo b^{2t}
   - Se $b^{2t} \equiv -1 \pmod{n}$ allora n è un possibile primo.
   - Se $b^{2t} \equiv 1 \pmod{n}$ allora n non è primo.
   - Altrimenti calcolo $b^{2^2 t}$.
5. Itero il procedimento calcolando $b^{2^k t}$ con $k \leqslant s-1$ fino a che non ottengo una risposta o fino a che non arrivo a $b^{2^{s-1} t}$.
   - Se $b^{2^{s-1} t} \equiv -1 \pmod{n}$ allora n è un possibile primo.
   - Altrimenti n non è primo.

Osservazione. Dopo m iterazioni positive del test, posso concludere che n è un falso primo con probabilità minore di $\frac{1}{2^{2m}}$.

3.4 PROBLEMI DI FATTORIZZAZIONE

Descrivendo RSA abbiamo già osservato come i problemi difficili che rendono sicuro l'algoritmo siano legati alla fattorizzazione di N. In particolare dimostreremo come fattorizzare N sia equivalente a calcolare $\varphi(N)$. Infine andremo a dimostrare l'equivalenza anche con il terzo problema, ovvero dati N ed e, trovare d tale che

$$(x^e)^d \equiv x \pmod{N}.$$

Proposizione 3.46 – Equivalenza tra fattorizzazione e calcolo della funzione di Eulero

Sia $N = p\,q$. Allora conosco la fattorizzazione di N se e soltanto se posso calcolare $\varphi(N)$.

Dimostrazione. Banale in quanto se conosco $N = p\,q$ posso calcolare $\varphi(N)$ tramite la definizione: $\Rightarrow)$

$$\varphi(N) = \varphi(p)\varphi(q) = (p-1)(q-1).$$

⇐) Supponiamo di conoscere il valore di $\varphi(N)$. Per ipotesi sappiamo che N è il prodotto di due primi p e q, per cui

$$\varphi(N) = (p-1)(q-1) = pq - (p+q) + 1 = N - (p+q) + 1.$$

A questo punto per trovare p, q possiamo risolvere il sistema

$$\begin{cases} pq = N \\ p + q = N - \varphi(N) + 1 \end{cases}$$

□

Proposizione 3.47 – Congruenza quadratica modulo N

Sia $N = pq$ con p, q primi. Allora la congruenza $x^2 \equiv a \pmod N$ ha quattro o nessuna soluzione, in particolare

$$\begin{cases} 4 \text{ soluzioni} & \text{se } \left(\frac{a}{p}\right) = 1 \text{ e } \left(\frac{a}{q}\right) = 1 \\ 0 \text{ soluzioni} & \text{se } \left(\frac{a}{p}\right) = -1 \text{ o } \left(\frac{a}{q}\right) = -1 \end{cases}$$

Dimostrazione. Supponiamo che

$$\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = 1.$$

Allora $x^2 \equiv a \pmod p$ ha due soluzioni a_1, a_2 e $x^2 \equiv a \pmod q$ ha due soluzioni b_1, b_2. Le quattro soluzioni modulo N saranno quindi le soluzioni dei sistemi

$$\begin{cases} x \equiv a_1 \pmod p \\ x \equiv b_1 \pmod q \end{cases} \qquad \begin{cases} x \equiv a_1 \pmod p \\ x \equiv b_2 \pmod q \end{cases}$$

$$\begin{cases} x \equiv a_2 \pmod p \\ x \equiv b_1 \pmod q \end{cases} \qquad \begin{cases} x \equiv a_2 \pmod p \\ x \equiv b_2 \pmod q \end{cases}$$

□

Osservazione. In generale si dimostra che se $N = p_1^{\alpha_1} \cdot \ldots \cdot p_s^{\alpha_s}$, allora la congruenza $x^2 \equiv a \pmod N$ ha 2^s soluzioni se

$$\left(\frac{a}{p_1}\right) = \left(\frac{a}{p_2}\right) = \ldots = \left(\frac{a}{p_s}\right) = 1$$

e zero altrimenti.

Prendiamo sempre $N = pq$ e consideriamo $x^2 \equiv 1 \pmod N$. In questo caso avremo sempre

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

Per cui vi sono quattro soluzioni modulo N. In particolare ve ne sono due, dette *radici banali*, soluzioni dei sistemi

$$\begin{cases} x \equiv 1 \pmod p \\ x \equiv 1 \pmod q \end{cases} \implies x \equiv 1 \pmod N; \qquad \begin{cases} x \equiv -1 \pmod p \\ x \equiv -1 \pmod q \end{cases} \implies x \equiv -1 \pmod N$$

E altre due, le *radici non banali*, che risolvono i sistemi

$$\begin{cases} x \equiv 1 \pmod p \\ x \equiv -1 \pmod q \end{cases} \qquad \begin{cases} x \equiv -1 \pmod p \\ x \equiv 1 \pmod q \end{cases}$$

Esempio. Sia $N = 403 = 13 \cdot 31$. Le soluzioni di $x^2 \equiv 1 \pmod{403}$ sono

$$1, 92, 311, 402,$$

dove $1, 402$ sono le radici banali.

Supponiamo di conoscere le radici non banali senza conoscere la fattorizzazione. In particolare, se x è una radice non banale avremo

$$\begin{cases} x \equiv 1 \pmod{p} \\ x \equiv -1 \pmod{q} \end{cases} \implies p \mid x - 1$$

quindi, calcolando $(N, x-1)$ o $(N, x+1)$, otterremmo un fattore di N.

Osservazione. Quindi in Miller-Rabin, se nel calcolo di

$$a^t, a^{2t}, \ldots, a^{2^k t},$$

incontriamo un 1 che non sia in prima posizione e che non sia preceduto da -1, detto k l'esponente per cui ciò accade, avremo che

$$a^{2^{k-1} t}$$

è una radice non banale dell'unità. Pertanto possiamo concludere non solo che N non è primo, ma ne abbiamo persino trovato un fattore.

Esiste un algoritmo probabilistico che, dati N, e, d, produce radici non banali dell'unità modulo N. Tale algoritmo prende in input un g con $1 < g < N$ e ritorna una radice non banale dell'unità oppure fallisce. Descriviamone il funzionamento:

1. Prendo $g \in \mathbb{Z}_n$ e controllo che $(g, N) = 1$, altrimenti ho fattorizzato N e l'algoritmo si ferma.

2. Per ipotesi $e\,d \equiv 1 \pmod{\varphi(N)}$, pertanto $e\,d - 1 = k\,\varphi(N)$. Inoltre $g \in U(\mathbb{Z}_n)$, quindi applicando Fermat ottengo

$$g^{e\,d-1} = g^{k\,\varphi(N)} = (g^{\varphi(N)})^k \equiv 1 \pmod{N}$$

3. Scrivo $e\,d - 1 = 2^s t$ con t dispari ed eseguo una successione di esponenziazioni modulo N:

$$g^t, g^{2t}, g^{2^2 t}, \ldots, g^{2^s t},$$

dove sappiamo che $g^{2^s t} \equiv 1 \pmod{N}$.

4. Se $g^t \equiv 1 \pmod{N}$ l'algoritmo fallisce perché non posso trovare un ulteriore radice.

5. Se $g^{2^i t} \equiv 1$ ma $g^{2^{i-1} t} \equiv -1$ l'algoritmo fallisce perché ho trovato una radice banale.

6. Altrimenti esiste h tale che $g^{2^h t} \equiv 1 \pmod{N}$ ma $g^{2^{h-1} t} \not\equiv \pm 1 \pmod{N}$. In tal caso l'algoritmo restituisce

$$g^{2^{h-1} t}$$

come radice non banale dell'unità modulo N.

Osservazione. Si dimostra che la probabilità che l'algoritmo fallisca è minore di un mezzo.

Proposizione 3.48 – Equivalenza della fattorizzazione con il terzo problema

Sia $N = pq$ e sia e con $(e, \varphi(N)) = 1$. Allora conosco la fattorizzazione di N se e soltanto se posso trovare d tale che

$$(x^e)^d \equiv x \pmod{N}.$$

⇒) ⇐)

Dimostrazione. Banale, poiché se conosco la fattorizzazione di N posso calcolare $\varphi(N)$. Trovare d si riduce quindi al calcolo dell'inverso di e modulo $\varphi(N)$.
Conoscendo N, e, d possiamo applicare l'algoritmo precedente per trovare una radice non banale dell'unità modulo N. Tramite questa abbiamo visto come sia possibile fattorizzare N. □

Esempio. Siano $N = 403, e = 23$ e $d = 47$. Scriviamo

$$ed - 1 = 1800 = 2^3 \cdot 225.$$

Prendiamo $g = 2$ e cominciamo con il calcolo delle potenze:

$$2^{225} \equiv 187 \pmod{403}$$
$$2^{2\cdot 225} \equiv 311 \pmod{403}$$
$$2^{4\cdot 225} \equiv 1 \pmod{403}$$

quindi 311 è una radice non banale dell'unità modulo 403. Possiamo quindi fattorizzare 403 calcolando

$$(403, 311 - 1) = 31 \qquad \text{e} \qquad (403, 311 + 1) = 13.$$

3.5 ATTACCHI AD RSA

In questo paragrafo ci occuperemo di alcuni attacchi ad RSA che non coinvolgono la fattorizzazione del modulo.

Per prima cosa osserviamo come l'equivalenza fra la fattorizzazione e la conoscenza dell'esponente di decifratura possono esporre il crittosistema ad usi che lo rendono insicuro. Nelle applicazioni reali, esiste un superutente, ovvero un'autorità certificata anche detta *Trent*, che distribuisce le chiavi RSA agli utenti che ne fanno richiesta. Per risparmiare il Trent potrebbe decidere di fornire a tutti lo stesso modulo N variando solo gli esponenti di cifratura e decifratura e, d. Le varie chiavi sarebbero quindi del tipo

$$(N, e_A, d_A), (N, e_B, d_B), (N, e_C, d_C), \dots$$

Questo approccio rende del tutto insicuro il sistema, infatti ogni utente possiede e, d ed è pertanto in grado di fattorizzare N e violare tutti gli altri utenti con il suo stesso modulo.

Questo riutilizzo del modulo N è pericoloso anche nel caso in cui A, B siano due utenti "amici" che condividono il modulo. Supponiamo infatti che C debba mandare lo stesso messaggio m ad A e B. I messaggi cifrati saranno rispettivamente

$$c_A = m^{e_A} \pmod{N} \qquad \text{e} \qquad c_B = m^{e_B} \pmod{N}.$$

Ipotesi quasi sempre soddisfatta

Supponiamo che l'attaccante sia a conoscenza di queste informazioni. In tal caso è in grado di decifrare il messaggio se $(e_A, e_B) = 1$. Infatti

$$(e_A, e_B) = 1 \implies \exists\, s, t : 1 = s\,e_A + t\,e_B.$$

Ora l'attaccante conosce c_A, c_B, per cui calcola

$$c_A^s\, c_B^t = (m^{e_A})^s (m^{e_B})^t = m^{s\,e_A + t\,e_B} = m.$$

Quindi non bisogna mai utilizzare lo stesso modulo N per utenti diversi

3.5.1 ATTACCO DI WIENER

L'attacco di Wiener ad RSA è un *attacco ad esponente di decifratura piccolo*. Questo tipo di attacco funziona infatti nelle ipotesi in cui

$$\sqrt{6}d < \sqrt[4]{N} \qquad \text{e} \qquad q < p < 2q,$$

dove e, d sono gli esponenti di cifratura e decifratura e $N = p\,q$ è il modulo.

Per descrivere questo tipo di attacco è necessario qualche accenno alla teoria delle frazioni continue

Definizione 3.49 – Frazione continua

Sia $x \in \mathbb{R}$, se ne da la sua espressione in *frazione continua* come

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \dots}}} = [a_0; a_1, a_2, a_3, \dots]$$

dove gli a_i sono interi positivi.

Esempio. Scriviamo in frazione continua $\frac{214}{35}$. Per farlo applichiamo l'algoritmo di Euclide delle divisioni con il resto:

$$\begin{aligned} 214 &= 6 \cdot 35 + 4 \\ 35 &= 8 \cdot 4 + 3 \\ 4 &= 1 \cdot 3 + 1 \\ 3 &= 3 \cdot 1 + 0 \end{aligned}$$

a questo punto è sufficiente considerare i quozienti parziali per ottenere la scrittura in frazione continua:

$$\frac{214}{35} = 6 + \cfrac{1}{8 + \cfrac{1}{1 + \cfrac{1}{3}}} = [6; 8, 1, 3].$$

Proprietà 3.50. Sia $x \in \mathbb{Q}$, allora la scrittura in frazione continua di x ha un numero finito di termini.

Definizione 3.51 – Convergenti

Consideriamo $x \in \mathbb{Q}$ nella scrittura in frazione continua $[a_1; a_2, \dots, a_n]$. Se ne definisce il k-*esimo convergente* C_k troncando l'espressione in frazione continua al k-esimo posto, ovvero

$$C_k = [a_1; a_2, \dots, a_k] \qquad \text{con } k < n.$$

Osservazione. Per un generico razionale $\frac{m}{n}$ si hanno $k = \max\{L(n), L(m)\}$ convergenti, che è pari al numero di passi nell'algoritmo euclideo.

Osservazione. In generale si dimostra che, preso $x \in \mathbb{Q}$, si ha

$$C_1 \leqslant C_3 \leqslant \dots \leqslant C_{2h+1} \leqslant n \leqslant C_{2h} \leqslant \dots \leqslant C_4 \leqslant C_2.$$

Quindi i convergenti dispari approssimano n dal basso mentre quelli pari lo approssimano dall'alto. Inoltre, i convergenti sono sempre crescenti.

Esempio. Nell'esempio precedente $x = [6; 8, 1, 3]$, da cui

$$C_1 = 6; \qquad C_2 = 6 + \frac{1}{8} = \frac{49}{8}; \qquad C_3 = 6 + \frac{1}{8+1} = \frac{55}{9}; \qquad C_4 = \frac{214}{35} = x.$$

Teorema 3.52 – Distanza di un convergente dal razionale di partenza

Sia $x \in \mathbb{Q}$ e siano $p, q \in \mathbb{Z}$ non necessariamente primi. Allora

$$\left| x - \frac{p}{q} \right| < \frac{1}{2q^2} \implies \frac{p}{q} \text{ è un convergente per } x.$$

Proposizione 3.53 – Calcolo dei convergenti

Sia $x = [a_1; a_2, \ldots, a_n] \in \mathbb{Q}$. Detto $C_k = \frac{p_k}{q_k}$ il k-esimo convergente di x, con $(p_k, q_k) = 1$, avremo

$$\begin{pmatrix} a_1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a_2 & 1 \\ 1 & 0 \end{pmatrix} \cdot \ldots \cdot \begin{pmatrix} a_k & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} p_k & p_{k-1} \\ q_k & q_{k-1} \end{pmatrix}$$

Passiamo ora all'attacco vero e proprio. Ricordiamo che per ipotesi

$$\sqrt{6}d < \sqrt[4]{N} \qquad \text{e} \qquad q < p < 2q.$$

In quanto attaccanti conosciamo solo la chiave pubblica (N, e). Inoltre sappiamo che

$$e\,d \equiv 1 \pmod{\varphi(N)} \implies e\,d = 1 + k\,\varphi(N).$$

Vogliamo dimostrare che sotto le nostre ipotesi $\frac{k}{d}$ è un convergente di $\frac{e}{N}$. Per farlo sfruttiamo il teorema sui convergenti enunciato in precedenza. Dobbiamo quindi dimostrare che

$$\left| \frac{e}{N} - \frac{k}{d} \right| < \frac{1}{2d^2}$$

Procediamo con la stima:

$$\left| \frac{e}{N} - \frac{k}{d} \right| = \frac{|e\,d - k\,N|}{N\,d} = \frac{|e\,d - k\,\varphi(N) + k\,\varphi(N) - k\,N|}{N\,d},$$

Da $e\,d = 1 + k\,\varphi(N)$ segue $e\,d - k\,\varphi(N) = 1$, per cui

$$\left| \frac{e}{N} - \frac{k}{d} \right| = \frac{|e\,d - k\,\varphi(N) + k\,\varphi(N) - k\,N|}{N\,d} = \frac{|1 + k\,(\varphi(N) - N)|}{N\,d} = \frac{|-1 + k\,(N - \varphi(N))|}{N\,d}$$
$$< \frac{k\,|N - \varphi(N)|}{N\,d} = \frac{k\,(N - \varphi(N))}{N\,d},$$

ora

$$p < 2q \implies N - \varphi(N) = p\,q - (p-1)(q-1) = p + q - 1 < 3q - 1 < 3q,$$

da cui

$$\left| \frac{e}{N} - \frac{k}{d} \right| < \frac{K\,(N - \varphi(N))}{N\,d} < \frac{k\,3q}{N\,d};$$

inoltre

$$q < p \implies N = p\,q > q^2 \implies q < \sqrt{N},$$

quindi

$$\left|\frac{e}{N}-\frac{k}{d}\right| < \frac{k\,3q}{N\,d} < \frac{3k\sqrt{N}}{N\,d} = \frac{3k}{d\sqrt{N}}.$$

Infine $e\,d - k\,\varphi(N) = 1$ ci dice che $e < \varphi(N) \implies k < d$, quindi

$$\left|\frac{e}{N}-\frac{k}{d}\right| < \frac{3k}{d\sqrt{N}} < \frac{3d}{d\sqrt{N}} = \frac{3}{\sqrt{N}}.$$

Ricordando che, per ipotesi, $\sqrt{6}d < \sqrt[4]{N}$, avremo $6d^2 < \sqrt{N}$. Quindi

$$\left|\frac{e}{N}-\frac{k}{d}\right| < \frac{3}{\sqrt{N}} < \frac{3}{6d^2} = \frac{1}{2d^2}.$$

Quindi, per il teorema precedente, $\frac{k}{d}$ è un convergente di $\frac{e}{N}$. Possiamo quindi sviluppare $\frac{e}{N}$ in frazioni continue e controllare tutti i convergenti che hanno la caratteristica della coppia (k, d).

Osservazione. Il numero di convergenti di $\frac{n}{m}$ è pari al numeri di passi dell'algoritmo euclideo, il quale è minore della lunghezza massima fra n e m. I convergenti sono pertanto in numero di $\mathcal{O}(L(N))$.

Per trovare il convergente corretto dobbiamo cercare una coppia (k, d) che sia plausibile. Non potremmo quindi accettare, ad esempio, d pari. Tramite la conoscenza di N, e calcoliamo

$$\varphi(N) = \frac{ed-1}{k},$$

che deve essere intero. Tramite N e il possibile $\varphi(N)$ possiamo fattorizzare N e verificare quindi se la coppia (k, d) è corretta. Sappiamo infatti che

$$\begin{cases} p\,q = N \\ p + q = N - \varphi(N) + 1 \end{cases}$$

quindi p, q si trovano risolvendo l'equazione

$$x^2 - (p+q)\,x + p\,q.$$

Esempio. Supponiamo che la chiave pubblica sia $N = 834443$ e $e = 499565$. Affinché l'attacco abbia successo si deve avere $\sqrt{6}d < \sqrt[4]{N}$. Ora

$$30 < \sqrt[4]{N} < 31,$$

quindi non considereremo eventuali $d > 31$ nel calcolo dei convergenti. Sviluppiamo $\frac{e}{N}$ in frazione continua e calcoliamone i convergenti:

$$\begin{aligned} 499565 &= 0 \cdot 834443 + 499565 &&\implies C_1 = 0 \\ 834443 &= 1 \cdot 499565 + 334878 &&\implies C_2 = 1 \\ 499565 &= 1 \cdot 334878 + 164685 &&\implies C_3 = \frac{1}{2} \\ 334878 &= 2 \cdot 164685 + 5504 &&\implies C_4 = \frac{3}{5} \\ 164685 &= 29 \cdot 5504 + 5071 &&\implies C_5 = \frac{89}{147} \end{aligned}$$

Da C_5 in poi i convergenti non sono più accettabili in quanto avrebbero $d > 31$. Inoltre C_1, C_2 sono chiaramente inaccettabili e C_2 non è accettabile poiché d è necessariamente dispari. L'unica possibilità si ha quindi con la coppia $k = 3, d = 5$. In

particolare

$$\varphi(N) = \frac{e\,d - 1}{k} = 832608$$

che è plausibile. Impostiamo la seguente equazione di secondo grado per il calcolo di p, q:

$$x^2 - (p+q)\,x + p\,q = 0 \iff x^2 - 1836x + 834443 = 0$$

da cui

$$p, q = \frac{1836 \pm \sqrt{1836^2 - 4 \cdot 834443}}{2} = \frac{1836 \pm 182}{2} \implies \begin{cases} p = 1009 \\ q = 827 \end{cases}$$

Che verificano la fattorizzazione di N.

3.5.2 BROADCAST ATTACK

Nel paragrafo precedente abbiamo visto come una scelta particolare dell'esponente di decifratura, renda possibile un attacco ad RSA. In questo paragrafo vedremo che lo stesso accade per scelte dell'esponente di cifratura molto piccole. Questo attacco è particolarmente interessante poiché nelle prime implementazioni di RSA si sceglieva sempre $e = 3$. Questo approccio, ora superato, presenta un evidente problema nella cifratura

$$x \longmapsto x^3 \pmod N$$

infatti, se x è sufficientemente piccolo, si può avere che $x^3 < N$ per cui non vi è nessuna riduzione modulare e il messaggio può essere estratto tramite una semplice radice cubica.

D'altronde un esponente di cifratura molto piccolo espone il crittosistema ad un vero e proprio attacco, detto *broadcast attack*. Supponiamo di voler mandare lo stesso messaggio a k utenti distinti con moduli RSA $N_1, \ldots, N_k$ diversi ma con lo stesso esponente di cifratura $e_1 = \ldots = e_k = k$. Otterremo quindi k testi cifrati:

$$\begin{cases} x^k \equiv y_1 \pmod{N_1} \\ x^k \equiv y_2 \pmod{N_2} \\ \vdots \\ x^k \equiv y_k \pmod{N_k} \end{cases}$$

Da attaccanti siamo interessati a studiare il sistema

$$\begin{cases} z \equiv y_1 \pmod{N_1} \\ z \equiv y_2 \pmod{N_2} \\ \vdots \\ z \equiv y_k \pmod{N_k} \end{cases}$$

Per il teorema cinese di resti, vi è un'unica soluzioni modulo $N_1 \cdot \ldots \cdot N_k$ e tale soluzione è proprio x^k come numero intero. Infatti

$$x < N_1, \ldots, N_k \implies x^k < N_1 \cdot \ldots \cdot N_k.$$

Pertanto possiamo trovare il messaggio x estraendo la k-esima radice da x^k.

Chiaramente questo attacco ha successo tanto più è piccolo l'esponente di cifratura k.

Esempio. Supponiamo che tre utenti A_1, A_2, A_3 ricevano lo stesso messaggio $m = 1500$ e supponiamo che le chiavi pubbliche degli utenti siano rispettivamente

$$(1711, 3); \qquad (1643, 3); \qquad (1739, 3).$$

Da attaccante intercettiamo i tre messaggi cifrati

$$y_1 = 1170; \qquad y_2 = 333; \qquad y_3 = 970.$$

Possiamo quindi impostare il sistema

$$\begin{cases} x \equiv 1170 \pmod{N_1} \\ x \equiv 333 \pmod{N_2} \\ x \equiv 970 \pmod{N_3} \end{cases}$$

Il sistema ha un'unica soluzione modulo $N_1N_2N_3$:

$$3375000000 = 1500^3$$

che ci fornisce il messaggio.

Ad oggi la scelta consigliata per e è $2^{16} + 1 = 65537$. La forma $2^n + 1$ comporta un'applicazione molto semplice dell'algoritmo delle esponenziazioni modulari, così da rendere più efficiente la cifratura.

Osservazione. Una scelta casuale di e porta a circa 1000 moltiplicazioni durante l'esponenziazione modulare contro le 2 della scelta $2^n + 1$.

3.6 CRITTOSISTEMA DI RABIN

Il crittosistema che analizzeremo in questo paragrafo è basato, come nel caso di RSA, sul problema della fattorizzazione. La particolarità che ne rende interessante lo studio è che, a differenza di RSA, la decifratura è del tutto equivalente alla fattorizzazione

Formalmente il crittosistema di Rabin ha le seguenti caratteristiche:

- $P = C = \mathbb{Z}_N$ con $N = p\,q$ dove p, q sono primi tali che $p, q \equiv 3 \pmod 4$.
- Lo spazio delle chiavi è dato da
$$K = (N, p, q),$$
dove N è la chiave pubblica mentre p, q sono quella privata
- Fissata la chiave k si ha
$$e_k(x) = x^2 \pmod N \qquad \text{e} \qquad d_k(y) = \sqrt{y} \pmod N$$

Questa funzione di cifratura ha un evidente problema, non è iniettiva. Infatti, abbiamo visto in precedenza che, se $N = p\,q$, la congruenza $x^2 \equiv y \pmod N$ ha zero oppure quattro soluzioni.

Per decifrare, una volta ricevuto y, per prima cosa si controlla che

$$\left(\frac{y}{p}\right)_L = \left(\frac{y}{q}\right)_L = 1,$$

se questo è falso il messaggio è stato cifrato male. Dopo si risolve il sistema

$$\begin{cases} x^2 \equiv y \pmod p \\ x^2 \equiv y \pmod q \end{cases}$$

dal momento che $p, q \equiv 3 \pmod 4$ la risoluzione di tali congruenze è particolarmente semplice. Per tale proprietà è lecito calcolare

$$z_p = y^{\frac{p+1}{4}} \pmod p \qquad \text{e} \qquad z_q = y^{\frac{q+1}{4}} \pmod q.$$

Ora

$$z_p^2 = y^{\frac{p+1}{2}} = y\, y^{\frac{p-1}{2}} \equiv y \pmod{p}$$

e analogamente $z_q^2 \equiv y \pmod{q}$. Per cui ho trovato le quattro radici $\pm z_p$ e $\pm z_q$. A questo punto per trovare le soluzioni complessive del sistema devo risolvere quattro sistemi di congruenze:

$$\begin{cases} w \equiv \pm z_p \pmod{p} \\ w \equiv \pm z_q \pmod{q} \end{cases} \qquad \text{e} \qquad \begin{cases} w \equiv \pm z_p \pmod{p} \\ w \equiv \mp z_q \pmod{q} \end{cases}$$

Anche in questo caso vi è un modo più immediato per il calcolo delle soluzioni. Calcolando l'identità di Bezout $1 = s\,p + t\,q$, si ottengono le 4 soluzioni tramite

$$\pm w_1 = \pm(s\,p\,z_q + t\,q\,z_p) \qquad \text{e} \qquad \pm w_2 = \pm(s\,p\,z_q - t\,q\,z_p).$$

Per ovviare al problema che ad ogni testo cifrato corrispondono quattro testi in chiaro si utilizzano delle tecniche di *ridondanza*. Di fatto si copia una porzione del messaggio e la si ripete in fondo allo stesso, così da rendere evidente quale testo in chiaro è quello corretto. Si può ad esempio decidere, per messaggi di k bit, di inserire il messaggio nei primi $2/3$ bit, mentre l'ultima parte del testo conterrà una ripetizione degli ultimi bit del messaggio.

Esempio. Sia $N = 84281 = 271 \cdot 311$. Poiché $L(N) = 17$ il nostro messaggio dovrà essere di 16 bit. Scegliamo quindi di inserire il nostro testo nei primi 10 bit e di ripeterlo negli ultimi 6. Quindi per inviare $m = 1001111001_2$ dovremo cifrare il messaggio

$$\bar{m} = 1001111001\underbrace{111001}_{2}$$

che in decimale corrisponde ad $x = 40569$. Il testo cifrato sarà pertanto

$$y = x^2 \equiv 4393 \pmod{N}.$$

Per decifrare, una volta verificato che

$$\left(\frac{4393}{271}\right)_L = \left(\frac{4393}{311}\right)_L = 1,$$

si deve risolvere il sistema

$$\begin{cases} z^2 \equiv 4393 \pmod{271} \\ z^2 \equiv 4393 \pmod{311} \end{cases}$$

che come sappiamo si ottengono da

$$z_p = 4393^{68} \equiv 81 \pmod{p} \qquad \text{e} \qquad z_q = 4393^{78} \equiv 138 \pmod{311}$$

Tramite l'identità di Bezout $1 = -70 \cdot 271 + 61 \cdot 311$, troviamo

$$w_1 = 61 \cdot 311 \cdot 81 - 70 \cdot 271 \cdot 138 \qquad e w_2 = 61 \cdot 311 \cdot 81 + 70 \cdot 271 \cdot 138$$

che ci forniscono le quattro soluzioni del sistema

$$79755; \qquad 4526; \qquad 40569; \qquad 43712.$$

Passando in binario si osserva immediatamente che solamente 40569 esibisce una ridondanza, ed è pertanto il messaggio corretto.

Come già accennato, l'interesse teorico del crittosistema di Rabin deriva dal fatto che attaccarlo con successo è del tutto equivalente alla fattorizzazione di N. Chiaramente, come per RSA, se conosco la fattorizzazione posso decifrare, dobbiamo quindi mostrare il viceversa. Supponiamo quindi di saper decifrare un testo cifrato di Rabin, questo significa che sono in

grado di estrarre una radice quadrata modulo N. Prendiamo quindi un r e calcoliamo $y = r^2 \pmod N$. Adesso sfruttiamo la nostra capacità di estrarre una radice quadrata. Se otteniamo $\pm r$ non possiamo ottenere ulteriori informazioni e dobbiamo cambiare base; se invece otteniamo una delle altre due radici $\pm s$ avrò la seguente informazione

$$r \not\equiv s \pmod N \qquad \text{ma} \qquad r^2 \equiv s^2 \pmod N.$$

Pertanto

$$r^2 - s^2 \equiv 0 \pmod N \implies (r-s)(r+s) \equiv 0 \pmod N \implies (r-s)(r+s) = kN.$$

Per cui, calcolando $(r+s, N)$ o $(r-s, N)$, otteniamo la fattorizzazione di N.

3.7 ALGORITMI DI FATTORIZZAZIONE

Un algoritmo di fattorizzazione considera tipicamente interi N dispari che non siano una potenza perfetta. Lo scopo di tali algoritmi, in generale, non è quello di fattorizzare completamente N, bensì di spezzarlo in maniera non banale, ovvero di scrivere $N \neq a^k$

$$N = n_1 n2 \qquad \text{con } 1 < n_1, n_2 < N.$$

Nel caso del modulo RSA di fatto spezzare è equivalente a fattorizzare N.

Distinguiamo due tipi di algoritmi di fattorizzazione: "speciali" e "generali". I primi operano quando N soddisfa determinate ipotesi e saranno, in questi particolari casi, molto efficienti; i secondi si applicano ad N qualsiasi. Nella nostra trattazione descriveremo perlopiù algoritmi speciali.

3.7.1 ALGORITMO P−1 DI POLLARD

La descrizione di questo algoritmo avverrà supponendo che $N = pq$. Ipotizziamo di conoscere un intero L tale che

$$p - 1 \mid L.$$

Scegliamo quindi $a < N$ e assumiamo che $(a, N) \neq 1$, altrimenti abbiamo già spezzato N. Ora $p - 1 \mid L \implies L = k(p-1)$, quindi

$$a^L = (a^{p-1})^k \equiv 1 \pmod p \implies p \mid a^L - 1.$$

Pertanto $(a^L - 1, N)$ ci fornisce p oppure N. Nel primo caso abbiamo spezzato N, altrimenti anche $q \mid a^L - 1$ e non è possibile concludere nulla.

Dal momento che la conoscenza di L è solo ipotetica, prenderemo un intero prodotto di molti fattori per massimizzare la probabilità di ottenere un numero con la proprietà cercata. La scelta migliore risulta pertanto essere un fattoriale.

> *Osservazione.* Vedremo infatti che questo algoritmo ha buone probabilità di successo nel caso in cui $p - 1$ abbia molti fattori piccoli

Scegliamo quindi $L = r!$ e descriviamo formalmente l'algoritmo:

Precondition: $(a, N) = 1, B < N$

```
r ← 2
d ← 1
while d = 1, r < B do
    d ← (a^{r!} − 1, N)
    if d ≠ 1 then
        return d
    r ← r + 1
```

Osserviamo che non è necessario calcolare $a^{r!} - 1$ ad ogni iterazione. Infatti

$$a^{r!} = a^{(r-1)!\,r} = \left(a^{(r-1)!}\right)^r,$$

quindi ad ogni passo devo calcolare una singola esponenziazione. Osserviamo inoltre che, in generale

$$(a, N) = \left(a \pmod N, N\right),$$

per cui l'esponenziazione è modulare. Il singolo passo si ricuce pertanto al calcolo di un'esponenziazione modulare e ad un MCD. La complessità di ogni iterazione è pertanto polinomiale, mentre quella complessiva dipende dalla relazione di B con N. In particolare, detto $k = L(N)$, la complessità sarà

$$B\left(L(B)\,k^2 + k^3\right),$$

dove il primo addendo è la complessità dell'esponenziazione modulare, mentre il secondo quella dell'MCD. Chiaramente se B è confrontabile con p, la complessità diventa esponenziale e non è più conveniente dei un algoritmo di divisioni successive.

L'algoritmo avrà successo nell'ipotesi in cui, scritto $p-1 = q_1^{\alpha_1} \cdot \ldots \cdot q_k^{\alpha_k}$, si abbia $q_i^{\alpha_i} < B$ per ogni i. In tal caso avremo infatti

$$p - 1 \mid B!.$$

Questa condizione rende l'algoritmo del tipo speciale.

Osservazione. L'algoritmo è chiaramente generale se non si pongono limiti alla grandezza di B.

Esempio. Consideriamo $N = 10001$, $a = 2$, $B = 10$ e applichiamo l'algoritmo:

1. $r = 2$:
$$a^{2!} = 2^2 = 4 \qquad \text{e} \qquad (4 - 1, N) = 1.$$
2. $r = 3$:
$$a^{3!} = 4^3 = 64 \qquad \text{e} \qquad (64 - 1, N) = 1.$$
3. $r = 4$:
$$a^{4!} = 64^2 \equiv 5539 \pmod N \qquad \text{e} \qquad (5539 - 1, N) = 1.$$
4. $r = 5$:
$$a^{5!} = 5539^5 \equiv 7746 \pmod N \qquad \text{e} \qquad (7746 - 1, N) = 1.$$
5. $r = 6$:
$$a^{6!} = 7746^6 \equiv 1169 \pmod N \qquad \text{e} \qquad (7746 - 1, N) = 73.$$

L'algoritmo restituisce pertanto 73 che ci permette di spezzare N:

$$N = 73 * 137.$$

Nella scelta dei moduli RSA bisogna pertanto evitare primi p tali che $p-1$ ha solo fattori primi piccoli. Primi adeguati sono i cosiddetti primi *sicuri*.

Definizione 3.54 – Primo sicuro

Un numero primo p si dice *forte* se è esprimibile nella forma

$$p = 2q + 1,$$

dove q è un altro numero primo.

Notazione. Il primo q si dice primo di *Germain*, dalla matematica Sophie Germain.

3.7.2 ALGORITMO RHO DI POLLARD

Sia $N = pq$. Ipotizziamo di conoscere due interi x e x' tali che

$$x \not\equiv x' \pmod N \qquad \text{e} \qquad x \equiv x' \pmod p.$$

Possiamo quindi calcolare $(x - x', N)$ per ottenere p. Per trovare x, x' fisso un dato iniziale x_0 su cui costruisco una successione di interi

$$x_1, x_2, \dots, x_t \qquad \text{con } x_{i+1} = f(x_i) \pmod N,$$

dove $f\colon \mathbb{Z}_n \to \mathbb{Z}_n, x \mapsto x^2 + a$, con $a = 1$ tipicamente. Per ogni coppia nella successione calcolo $(x_i - x_j, N)$ fino ad ottenere un risultato diverso da 1.

In questa forma l'algoritmo è computazionalmente pesante per via delle numerose coppie presenti in un insieme anche di piccole dimensioni. La scelta delle coppie può essere affinate in modo da non renderla del tutto arbitraria. Osserviamo che se x_i, x_j sono una coppia che soddisfa le condizioni dell'algoritmo, allora anche x_{i+1}, x_{j+1} lo è, infatti

$$x_{i+1} - x_{j+1} = x_i^2 + 1 - x_j^2 - 1 = (x_i - x_j)(x_i + x_j),$$

per cui chiaramente

$$x_i \equiv x_j \pmod p \implies x_{i+1} \equiv x_{j+1} \pmod p$$

Questo ci dice che la nostra successione $x_1, \dots, x_i, \dots, x_j, \dots$ si ripete modulo p.

Osservazione. Il nome *rho* per l'algoritmo deriva proprio da questa caratteristica. Se si rappresenta graficamente la successione identificando gli elementi modulo p si ottiene una figura che ricorda la lettera ρ.

Per ridurre il numero di MCD da calcolare si utilizza il *metodo di Floyd*. Si definisce una successione di coppie (x_i, x_j) a partire dalla successione e si calcolerà l'MCD solamente su tali coppie. La successione è definita, definito un dato iniziale x_0, dalle leggi

$$\begin{cases} x_0 = x_0 \\ x_{i+1} = f(x_i) \pmod N \end{cases} \qquad \text{e} \qquad \begin{cases} y_0 = x_0 \\ y_{i+1} = f\big(f(y_i)\big) \pmod N \end{cases}$$

I primi elementi della successione saranno pertanto

$$(x_0, x_0), (x_1, x_2), (x_2, x_4), \dots$$

Possiamo quindi scrivere formalmente l'algoritmo, considerando $x_0 = 2$.

```
1: x ← 2;
2: y ← 2;
3: d ← 1
4: while d = 1 do
5:     x ← f(x);
6:     y ← f(f(x))
7:     d ← (x − y, N)
8: if d ≠ N then
9:     return N
10: else
11:     return Fallimento
```

Osserviamo che nell'algoritmo appena proposto non abbiamo effettivamente calcolato nessuna delle due successioni descritte in precedenza. Bensì abbiamo considerato un elemento alla volta di tali successioni procedendo iterativamente al calcolo dell'MCD.

Osservazione. Siano t, s sono gli indici della prima collisione, ovvero tali che

$$x_t \equiv x_s \pmod{p}.$$

Detta $l = s - t$ l'ampiezza del ciclo di ripetizione all'interno della successione, l'algoritmo di Floyd trova una collisione in al più $t + l$ passi.

Esempio. Consideriamo $N = 55, x_0 = 3, f(x) = x^2 + 1 \pmod{N}$. Applichiamo l'algoritmo calcolando esplicitamente la successione e la successione di coppie. I primi dieci elementi della successione saranno

$$10, 46, 27, 15, 6, 37, 50, 26, 17.$$

La successione di coppie sarà pertanto

$$(3,3), (10,46), (46,15), (27,37), (15,26).$$

Calcolando i vari MCD si trova

$$(27 - 37, N) = 5.$$

Osserviamo che passando modulo 5 si individua immediatamente una ripetizione all'interno della successione.

Non vi è una stima precisa della complessità computazionale dell'algoritmo rho. Una eventuale stima si baserebbe sul medesimo concetto del paradosso del compleanno, portando a dedurre che se $t \geqslant \sqrt[4]{N}$, la probabilità di successo è maggiore di $\frac{1}{2}$.

3.7.3 METODO DI FERMAT

Sia $N = p\,q$. Ipotizziamo di conoscere $x, y \in \mathbb{Z}$ tali che

$$N + y^2 = x^2.$$

Allora

$$N = x^2 - y^2 = (x - y)(x + y),$$

per cui se $x - y \neq 1$ abbiamo spezzato N. L'algoritmo di Fermat impone un limite superiore B e considera $y = 1, 2, \ldots, B$. Successivamente calcola $N + y^2$ verificando se è un quadrato perfetto, in tal caso riporta la fattorizzazione. Formalmente

Precondition: B<N
1: $y \leftarrow 1$
2: $x^2 \leftarrow N + y^2$
3: **while** $x^2 \neq \square$ **do**
4: $\quad y \leftarrow y + 1$
5: $\quad x^2 \leftarrow N + y^2$
6: **return** $x - y$

Se non viene imposto un limite B, l'algoritmo termina per al più

$$y = \frac{N-1}{2} \qquad \text{infatti } N + \left(\frac{N-1}{2}\right)^2 = \left(\frac{N+1}{2}\right)^2$$

Questo algoritmo è efficiente se $N = p\,q$ e $|p - q|$ è piccolo.

3.7.4 METODO DELLE BASI DI FATTORIZZAZIONE

I moderni algoritmi di fattorizzazione si basano sull'idea di applicare il metodo di Fermat modulo N. Si cerca quindi di trovare x, y tali che

$$x \not\equiv \pm y \pmod{N} \qquad \text{e} \qquad x^2 \equiv y^2 \pmod{N}.$$

In tal caso $N \mid x^2 - y^2 = (x-y)(x+y)$, per cui si può scomporre N calcolando

$$(N, x+y) \qquad \text{e} \qquad (N, x-y).$$

Una tecnica per la ricerca di tali x, y si chiama delle *basi di fattorizzazione*.

Definizione 3.55 – Interi B-smooth

Un intero n si dice B-smooth se nessuno dei suoi fattori primi è maggiore di B. Ovvero

$$n = p_1^{\alpha_1} \cdot \ldots \cdot p_s^{\alpha_s} \implies p_i \leqslant B \; \forall \; i.$$

Notazione. Un intero B-smooth si dice anche B-numero.

Esempio. $n = 504$ è 7-smooth, infatti $504 = 2^3 \cdot 3^2 \cdot 7$.

Definizione 3.56 – B-vettore

Supponiamo che n sia un B-numero. Scriviamo la fattorizzazione di n facendo comparire tutti i primi minori o uguali a B con le rispettive potenze, supponendo che vi siano b primi minori di B, avremo

$$n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_b^{\alpha_b} \qquad \text{con } \alpha_i \geqslant 0.$$

Possiamo quindi associare ad n un B-vettore composto dai b esponenti della precedente fattorizzazione, ovvero

$$v_n = (\alpha_1, \alpha_2, \ldots, \alpha_b).$$

Esempio. Consideriamo l'intero dell'esempio precedente: $n = 2^3 \cdot 3^2 \cdot 7$. n è 7-smooth, per cui possiamo associargli un 7-vettore scrivendone la sua fattorizzazione completa:

$$n = 2^3 \cdot 3^2 \cdot 5^0 \cdot 7 \implies v_n = (3, 2, 0, 1).$$

Chiaramente la scelta di 7 per descrivere n è arbitraria, infatti n è B-smooth per ogni $B \geqslant 7$. Per $B > 7$ dovremmo considerare anche i primi $7 < p \leqslant B$, ad esempio se $B = 12$ avremo

$$n = 2^3 \cdot 3^2 \cdot 5^0 \cdot 7 \cdot 11^0 \implies v_n = (3, 2, 0, 1, 0).$$

L'idea delle basi di fattorizzazione è quella di trovare vari interi z tali che $z^2 \pmod{N}$ sia B-smooth rispetto ad un'appropriata scelta di B.

Nel prossimo esempio mostreremo il procedimento per sfruttare la conoscenza di tali z, giustificandone in seguito la scelta

Esempio. Consideriamo $N = 84923$ e supponiamo di voler cercare interi 7-smooth. Come detto supponiamo di aver trovato, giustificandolo in seguito, i seguenti interi

$z_1 = 513$ e $z_2 = 537$. Ora

$$z_1^2 \pmod N = 8400 = 2^4 \cdot 3 \cdot 5^2 \cdot 7 \qquad \text{e} \qquad z_2^2 \pmod N = 33600 = 2^6 \cdot 3 \cdot 5^2 \cdot 7$$

quindi entrambi sono, modulo N, interi 7-smooth. Consideriamo ora

$$x = z_1 \cdot z_2 \pmod N = 20712 \qquad \text{e} \qquad y^2 = z_1^2 \cdot z_2^2,$$

dove y^2 è un quadrato perfetto in quanto

$$z_1^2 \cdot z_2^2 = 2^{10} \cdot 3^2 \cdot 5^4 \cdot 7^2 = (2^5 \cdot 3 \cdot 5^2 \cdot 7)^2 = 16800^2.$$

Abbiamo quindi trovato x, y tali che $x \not\equiv \pm y \pmod N$ ma $x^2 \equiv y^2 \pmod N$. Possiamo quindi fattorizzare N calcolando

$$(N, 20712 - 16800) = 163 \qquad \text{e} \qquad (N, 20712 + 16800) = 521.$$

Osserviamo che nell'esempio z_1, z_2 sono tali che

$$v_{z_1} + v_{z_2} = (4,1,2,1) + (6,3,2,1) = (10,2,4,2) \equiv (0,0,0,0) \pmod 2.$$

Stiamo quindi cercando una relazione di dipendenza lineare sullo spazio vettoriale $\mathbb{Z}_2^4$.

In generale, avremo che vi sono b primi minori o uguali di B, quindi i B-vettori modulo 2 sono vettori dello spazio vettoriale $\mathbb{Z}_2^b$. Se scelgo $z_1, \ldots, z_c$ tali che z_i^2 modulo N è un B-numero, affinché si abbiamo sicuramente una relazione del tipo

$$a_1 v_{z_1} + a_2 v_{z_2} + \ldots + a_c v_{z_c} = (0,0,\ldots,0),$$

si deve avere $c \geqslant b+1$, poiché, come sappiamo, $b+1$ vettori sono dipendenti in uno spazio vettoriale di dimensione b.

L'algoritmo che ci apprestiamo a discutere, detto *Dixon's random square*, sceglie in maniera casuale, ma opportuna, i possibili z_i. Di fatto considera

$$z_{i,k} = \lfloor \sqrt{kN} \rfloor + i \qquad \text{con } k = 1, \ldots, l_k \text{ e } i = 1, \ldots, l_i.$$

Questa scelta fa in modo che $z_{i,k}^2$ sia piccolo modulo N e che, pertanto, il calcolo della sua fattorizzazione sia facile.

Esempio. Consideriamo $N = 32033$ e applichiamo l'algoritmo per trovare dei possibili z_i:

k	$x = \lfloor \sqrt{kN} \rfloor + 1$	$x^2 \pmod N$	Fattorizzazione di x^2
1	179	8	2^3
2	254	450	$2 \cdot 3^2 \cdot 5^2$
3	310	1	1
4	358	32	2^5

fingiamo di non aver trovato il terzo elemento che corrisponde ad una radice non banale dell'unità e che ci permetterebbe una facile fattorizzazione. Consideriamo z_1 e z_4 i cui quadrati sono 2-smooth e il cui prodotto costituisce un quadrato perfetto, avremo quindi

$$x = z_1 \cdot z_4 \pmod N = 16 \qquad \text{e} \qquad y^2 = 2^8 \implies y = 16,$$

questa scelta non è quindi accettabile in quanto $x \equiv y \pmod N$. Consideriamo quindi z_1 e z_2 i cui quadrati sono 5-smooth. Avremo

$$x = z_1 \cdot z_2 \pmod N = 13433 \qquad \text{e} \qquad y^2 = 2^4 \cdot 3^2 \cdot 5^2 \implies y = 60.$$

Possiamo quindi fattorizzare calcolando

$$(N, 13433 - 60) = 311 \qquad \text{e} \qquad (N, 13433 + 60) = 103.$$

La complessità di questo algoritmo si stima essere sotto-esponenziale, in questo corso non ci addentreremo nel calcolo.

4 IL PROBLEMA DEL LOGARITMO DISCRETO

4.1 INTRODUZIONE

Cominciamo col richiamare alcune definizioni e proprietà dei gruppi ciclici, per poi introdurre il problema del logaritmo discreto.

Definizione 4.1 – Gruppo ciclico

Un gruppo G si dice ciclico se esiste $g \in G$ tale che

$$G = \langle g \rangle.$$

Esempio. $U(\mathbb{Z}_9) = \langle 2 \rangle$ è un gruppo ciclico, mentre $U(\mathbb{Z}_8)$ non lo è.

Proprietà 4.2. Il gruppo $U(\mathbb{Z}_p)$ degli invertibili di $\mathbb{Z}_p$, con p primo, è ciclico.

Proprietà 4.3. Se $\mathbb{F}_{p^m}$ è un campo finito, allora il suo gruppo moltiplicativo

$$(\mathbb{F}_{p^m} \setminus \{0\}, \cdot),$$

è un gruppo ciclico.

Definizione 4.4 – Problema del logaritmo discreto

Sia G ciclico di ordine n e sia g un generatore di G. Dato $y \in G, y \neq 1$, il *problema del logaritmo discreto* è quello di determinare l'unico x tale che

$$g^x = y \qquad \text{con } 1 \leqslant x \leqslant n-1.$$

Notazione. L'intero x si chiama logaritmo discreto di y in base g e si scrive

$$x = \log_g y.$$

Tipicamente lavoreremo in $\mathbb{Z}_p^*$ o, più in generale, in $\mathbb{F}_q^*$ con $q = p^m$ e p primo. Vedremo come, dati g generatore di $\mathbb{Z}_p^*$ e x tale che $1 \leqslant x \leqslant p-1$, calcolare $y = g^x$ è computazionalmente facile (si tratta di applicare l'algoritmo square and multiply); viceversa, si ritiene che, dati g generatore di $\mathbb{Z}_p^*$ e $y \in \mathbb{Z}_p^*$, determinare $x = \log_g y$ sia computazionalmente difficile.

Osservazione. Quest'ultima affermazione è vera per opportune scelte di p, in particolare p deve essere molto grande.

4.2 SCAMBIO DELLA CHIAVE DI DIFFIE–HELLMAN

Un protocollo di scambio della chiave serve affinché due interlocutori, che inizialmente non condividono nessuna informazione segreta, possano, al termine del protocollo, condividere una chiave segreta. Chiaramente il procedimento è studiato in modo tale, che un eventuale ascoltatore non ottenga nessuna informazione sulla chiave.

Come al solito faremo riferimento ai due interlocutori con i nomi di Alice e Bob, o, più sinteticamente, con A e B. Descriviamo il protocollo:

- A e B scelgono pubblicamente un primo p e un elemento primitivo $g \pmod p$.
- A sceglie casualmente $a \in \{2, \dots, p-2\}$ e calcola $g^a \pmod p$, inviando il risultato a B.
- B sceglie casualmente $b \in \{2, \dots, p-2\}$ e calcola $g^b \pmod p$, inviando il risultato ad A.
- A calcola $(g^b)^a \pmod p$.
- B calcola $(g^a)^b \pmod p$.
- La chiave segreta è
$$k = g^{a\,b}$$

La sicurezza del protocollo si basa sul problema del logaritmo discreto, anche se non è del tutto equivalente. Chiaramente se l'attaccante è in grado di risolvere il logaritmo discreto allora è in grado di ricavare la chiave segreta. D'altronde otterrebbe lo stesso risultato se fosse in grado di ottenere $g^{a\,b}$ conoscendo g^a e g^b.

4.3 CENNI SUI CAMPI FINITI

Proprietà 4.5. Sia F un campo finito, allora $|F| = p^m$ con p primo.

Proprietà 4.6. Per ogni p primo e per ogni $m \geqslant 1$, esiste un unico campo finito F con p^m elementi. Tale campo si denota con $\mathbb{F}_{p^m}$.

Osservazione. Per $m = 1$ si ha $\mathbb{F}_p = \mathbb{Z}_p$.

In generale $\mathbb{F}_{p^m}$ si costruisce quozientando l'anello dei polinomi $\mathbb{Z}_p[X]$ con l'ideale generato da un polinomio irriducibile $f(X) \in \mathbb{Z}_p[X]$ di grado m, ovvero

$$\mathbb{F}_{p^m} = \frac{\mathbb{Z}_p[X]}{(f(X))} \qquad \text{con } \deg f = m.$$

Esempio. Il polinomio $f(X) = X^3 + X + 1$ è irriducibile in $\mathbb{Z}_2[X]$, per cui possiamo costruire il campo finito
$$\mathbb{F}_8 = \frac{\mathbb{Z}_2[X]}{(f(X))}.$$

Con metodi analoghi a quelli utilizzati nei capitoli precedenti, possiamo stimare la complessità computazionale in $\mathbb{Z}_p$ e $\mathbb{F}_q$ con $q = p^m$. In $\mathbb{Z}_p$, detto $k = L(p)$, avremo

- Somma e differenza: $\mathcal{O}(k)$.
- Prodotto: $\mathcal{O}(k^2)$.

- Calcolo inverso moltiplicativo: $\mathcal{O}(k^3)$.
- Esponenziazione modulare: $\mathcal{O}(k^3)$.

In $\mathbb{F}_q$, detto $h = L(q) = m\,L(p)$, avremo

- Somma e differenza: $\mathcal{O}(h)$.
- Prodotto: $\mathcal{O}(h^2)$.
- Calcolo inverso moltiplicativo: $\mathcal{O}(h^3)$.
- Esponenziazione: $\mathcal{O}\big(h^3 L(e)\big)$.

Definizione 4.7 – Polinomio primitivo

Si definisce *polinomio primitivo* il polinomio minimo di un elemento primitivo di un campo finito $\mathbb{F}_q$ con $q = p^m$.

Osservazione. In particolare un polinomio primitivo sarà un polinomio irriducibile in $\mathbb{Z}_p[X]$ di grado m.

Esempio. Consideriamo il campo finito dell'esempio precedente

$$\mathbb{F}_8 = \frac{\mathbb{Z}_2[X]}{(X^3 + X + 1)}.$$

Come vedremo nel prossimo paragrafo, $\mathbb{F}_8^*$ è ciclico e inoltre $|\mathbb{F}_8^*| = 7$ è primo, per cui ogni elemento distinto dall'identità è un generatore. Ad esempio X lo è. Pertanto $X^3 + X + 1 \in \mathbb{Z}_2[X]$ è un polinomio primitivo.

4.4 ALCUNE PROPRIETÀ DEI GRUPPI CICLICI

Sia G un gruppo moltiplicativo ciclico di ordine n. In particolare esiste $g \in G$ generatore di G. Quindi

$$\operatorname{ord}(g) = n \qquad \text{e} \qquad G = \langle g \rangle = \left\{ g, g^2, \ldots, g^{n-1}, g^n = 1 \right\}.$$

Proprietà 4.8. Sia g un generatore di G, allora $a = g^i$ è un altro generatore di G se e soltanto se

$$(i, n) = 1.$$

Osservazione. In particolare segue che G ha $\varphi(n)$ generatori.

Proprietà 4.9. Un gruppo ciclico ha $\varphi(d)$ elementi di ordine d per ogni $d \mid n$.

Osservazione. Da ciò segue che G ha sottogruppi di ordine n per ogni $d \mid n$.

Proprietà 4.10. La cardinalità di G soddisfa la seguente identità:

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

Lemma 4.11. Sia G un gruppo finito moltiplicativo con $|G| = n$. Supponiamo che per ogni $d \mid n$ si abbia che

$$\#\left\{ x \in G \;\middle|\; x^d = 1 \right\} \leqslant d.$$

Allora G è ciclico.

Dimostrazione. Supponiamo che G sia il gruppo in ipotesi. Fissiamo $d \mid n$ e definiamo G_d l'insieme degli elementi di ordine d in G. Se $G_d \neq \emptyset$ allora esiste $y \in G_d$. Consideriamo il gruppo generato da y, per costruzione avremo che $|\langle y \rangle| = d$, da cui

chiaramente G_d può essere vuoto

$$\langle y \rangle \subseteq \left\{ x \in G \;\middle|\; x^d = 1 \right\} \implies \langle y \rangle = \left\{ x \in G \;\middle|\; x^d = 1 \right\},$$

poiché, per ipotesi,

$$d = |\langle y \rangle| \leqslant \left|\left\{ x \in G \;\middle|\; x^d = 1 \right\}\right| \leqslant d.$$

Segue che G_d è l'insieme dei generatori di $\langle y \rangle$, per cui $|G_d| = \varphi(d)$. Quindi per ogni $d \mid n$, si ha

$$|G_d| = \begin{cases} \varphi(d) & \text{se } G_d \neq \emptyset \\ 0 & \text{altrimenti} \end{cases} \implies |G_d| \leqslant \varphi(d).$$

Ora

$$n = |G| = \sum_{d|n} |G_d| \leqslant \sum_{d|n} \varphi(d) = n \implies |G_d| = \varphi(d) \,\forall\, d \mid n.$$

Da cui $G_d \neq \emptyset$ per ogni $d \mid n$ e, in particolare, $G_n \neq \emptyset$; ovvero c'è almeno un elemento di ordine n in G e pertanto G è ciclico. □

Teorema 4.12 – Gruppo moltiplicativo di un campo finito è ciclico

Sia $\mathbb{F}_q$ un campo finito. Allora $\mathbb{F}_q^*$ è un gruppo ciclico.

Dimostrazione. Applichiamo il lemma precedente. Osserviamo che l'insieme

$$\left\{ x \in \mathbb{F}_q \;\middle|\; x^d = 1 \right\} = \left\{ x \in \mathbb{F}_q \;\middle|\; x \text{ radice di } x^d - 1 = 0 \right\}$$

e in un campo un polinomio di grado d ha al più d radici. Per cui l'ipotesi del teorema è soddisfatta, da cui la tesi. □

4.5 RICERCA DI RADICI PRIMITIVE

Consideriamo $\mathbb{F}_q$ con $q = p^m, m \geqslant 1$. Per definizione $|F_q^*| = q - 1$, per cui $\mathbb{F}_q^*$ ha $\varphi(q-1)$ radici primitive. Da ciò segue che scegliendo casualmente un elemento di F_q^*, la probabilità che questo sia una radice primitiva è

$$\frac{\varphi(q-1)}{q-1}.$$

Per stimare tale probabilità è necessario conoscere la fattorizzazione di $q - 1$. Un caso particolarmente favorevole si ha per i primi di Germain, ovvero per i primi p tali che $q =$

$2p+1$ è primo. Se scegliamo q e consideriamo $\mathbb{Z}_q^*$, avremo $|\mathbb{Z}_q^*| = 2p$, per cui vi sono $\varphi(2p) = p-1$ radici primitive. In particolare, la probabilità di individuare casualmente una radice primitiva è

$$\frac{\varphi(q-1)}{q-1} = \frac{p-1}{2p} \approx \frac{1}{2}.$$

Proposizione 4.13 – Radici primitive per primi di Germain

Consideriamo $\mathbb{F}_q$ con $q = 2p+1$ e p un primo di Germain. Se $x \in \{2, \dots, q-2\}$ allora x è una radice primitiva se e soltanto se

$$x^p \equiv -1 \pmod{q}.$$

Dimostrazione. Sappiamo che $|\mathbb{Z}_q^*| = 2p$, per cui ogni elemento distinto da ± 1 ha ordine p oppure $2p$. Inoltre, dal criterio di Eulero, sappiamo che

$$x^p = x^{\frac{q-1}{2}} \equiv \pm 1 \pmod{q}.$$

Per cui può accadere che $x^p \equiv 1$, ovvero x ha ordine p; oppure

$$x^p \equiv -1 \implies x^{2p} \equiv 1 \pmod{q},$$

ovvero x è una radice primitiva. □

Osservazione. Tramite il simbolo di Legendre è computazionalmente facile testare se $x^p \equiv -1 \pmod{q}$. Infatti sappiamo che

$$\left(\frac{x}{q}\right)_L = x^{\frac{q-1}{2}} \equiv \pm 1 \pmod{q}.$$

Proposizione 4.14 – Radici primitive per fattorizzazione nota

Consideriamo il campo finito $\mathbb{F}_q$ e supponiamo di conoscere la fattorizzazione $q-1 = p_1^{\alpha_1} \cdot \ldots \cdot p_s^{\alpha_s}$. Allora $g \in \mathbb{F}_q^*$ è una radice primitiva se e soltanto se

$$g^{\frac{q-1}{p_i}} \not\equiv 1 \pmod{q} \, \forall \, p_i \mid q-1.$$

Dimostrazione. Da fare. □

4.6 CRITTOSISTEMA DI ELGAMAL

Il crittosistema di Elgamal, sviluppato nel 1985, è un crittosistema a chiave pubblica che sfrutta il problema del logaritmo discreto. Scelto p primo e g un elemento primitivo modulo p, si ha:

- $P = \mathbb{Z}_p^*$ e $C = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$.
- Lo spazio delle chiavi è

$$K = \{ (p, g, a, \beta) \mid \beta \equiv g^a \pmod{p} \}.$$

Da questo insieme distinguiamo

- (p, g, β) la chiave pubblica.
- a la chiave privata.

Sia $k = (p, g, a, \beta) \in K$ una chiave. Supponiamo di avere $x \in P$, per criptare il mittente sceglie casualmente, e tenendo segreto, $h \in \{2, \dots, p-2\}$ e calcola

$$e_k(x, h) = (y_1, y_2) \qquad \text{dove} \begin{cases} y_1 = g^h \\ y_2 = x\,\beta^h \pmod p \end{cases}$$

Il destinatario riceve quindi una coppia (y_1, y_2), per decifrare sfrutta la sua conoscenza di a e calcola

$$d_k\big((y_1, y_2)\big) = y_2 (y_1^a)^{-1} \pmod p.$$

Osserviamo che, pur non conoscendo h, questo fornisce il messaggio. Infatti

$$y_2 (y_1^a)^{-1} = x\, b^h (g^{a\,h})^{-1} = x\, g^{a\,h} (g^{a\,h})^{-1} \equiv x \pmod p.$$

Osservazione. Il mittente sceglie un nuovo h ad ogni trasmissione, rendendo così il risultato della cifratura randomico anche per messaggi identici.

Osservazione. Come nello scambio della chiave di Diffie-Hellman, anche in questo procedimento non è necessario invertire la funzione

$$x \longmapsto g^x \pmod p.$$

Esempio. Prendiamo $p = 83, g = 2$ e la chiave privata $a = 30$. Avremo quindi $\beta = g^a \equiv 40 \pmod p$, da cui la chiave pubblica

$$(p, g, \beta) = (83, 2, 40).$$

Supponiamo che Alice voglia cifrare $x = 54$. Sceglie casualmente $h = 13$ e invia a Bob

$$(y_1, y_2) = (g^h, x\,\beta^h) \equiv (58, 71) \pmod p.$$

Per decifrare Bob calcola

$$(g^h)^a \equiv 9 \pmod p \qquad \text{da cui } (g^h)^{-a} = 9^{-1} \equiv 37 \pmod p.$$

Pertanto $x = y_2 y_1^{-a} = 71 \cdot 37 \equiv 54 \pmod{83}$.

Osservazione. Di fatto la cifratura $x \mapsto x\, g^{h\,a}$ è una moltiplicazione. Alternativamente si può usare un cifrario a blocchi (ad esempio AES) e cifrare

$$x \longmapsto e_{g^{h\,a}}(x).$$

Anche per il crittosistema di Elgamal, si utilizzano primi p con determinate caratteristiche. In particolare si usa p di almeno 2048 bit, dove $p - 1$ deve essere a fattorizzazione nota e con un fattore primo grande.

Come per lo scambio della chiave di Diffie-Hellman, anche il crittosistema di Elgamal può essere implementato utilizzando un gruppo ciclico arbitrario. Una scelta comune è il gruppo moltiplicativo di un gruppo finito $F_{p^m}^*$, in particolare con $p = 2$.

4.7 SICUREZZA DEL LOGARITMO DISCRETO

In questo paragrafo ci occuperemo di descrivere alcuni algoritmi, sia generali che speciali, per ricavare il logaritmo discreto. Tali algoritmi sono applicabili a qualsiasi gruppo G ciclico una

volta data un'opportuna struttura al gruppo; per questioni di semplicità gli esempi verranno presi in $G = \mathbb{Z}_p^*$.

Ricordiamo che dato g un generatore di G e preso $y \in G$ con $y \neq 1$, il problema è quello di determinare x tale che $g^x = y$. Il metodo più elementare per ricavare x è l'attacco a forza bruta. Tale metodo consiste nel calcolo di

$$g^i \qquad \text{per } i = 1, \dots, n-1, \text{con } n = |G|.$$

Chiaramente il numero di tentativi necessari per avere successo è dell'ordine $\mathcal{O}(n \approx 2^k)$, il che rende tale approccio paragonabile al metodo di divisioni successive per la fattorizzazione.

Osservazione. L'unico motivo pratico per cui potrebbe essere necessario un attacco a forza bruta è quello di catalogare interamente G tramite il generatore g.

4.7.1 ALGORITMO DI SHANKS

L'algoritmo di Shanks, anche detto algoritmo *baby-step giant-step*, è composto di due passi principali. Il primo è precomputabile poiché dipende solo dal gruppo su cui si opera e non da y.

Detto n l'ordine di G, definiamo $m = \lceil \sqrt{n} \rceil$. Procediamo con la descrizione dei passi algoritmo:

1. Si calcola (j, g^{mj}) per $j = 0, 1, \dots, m-1$ e si costruisce la lista L_1 di tali coppie ordinata rispetto alla seconda componente.
2. Si calcola $(i, y\, g^{-i})$ per $i = 0, 1, \dots, m-1$ e si costruisce la lista L_2 di tali coppie.
3. Si cercano due coppie $(j, a) \in L_1$ e $(i, a) \in L_2$ con la stessa seconda componente a. Se tale collisione esiste si ha

$$g^{mj} = y\, g^{-i} \implies y = g^{mj+i} \implies \log_g y = mj + i.$$

Osservazione. La lista L_1 viene ordinata poiché, durante il calcolo degli elementi di L_2, si può facilmente verificare un'eventuale collisione.

Esempio. Sia $G = \mathbb{Z}_p^*$ con $p = 37$. Prendiamo $g = 2$ e supponiamo di voler trovare $\log_g 28$. L'ordine di G è 36, da cui

$$m = \lceil \sqrt{36} \rceil = 6.$$

Applicando l'algoritmo otteniamo le liste L_1 e L_2:

$$L_1 = \{ (0,1), (4,10), (5,11), (2,26), (1,27), (3,36) \}$$
$$L_2 = \{ (0,28), (1,14), (2,7), (3,22), (4,11), (5,24) \}.$$

Vi è una collisione tra $(5,11) \in L_1$ e $(4,11) \in L_2$, pertanto

$$\log_g 28 = 34.$$

4.7.2 ALGORITMO DI POHLIG-HELLMAN

Questo algoritmo sfrutta la fattorizzazione dell'ordine di G per la ricerca del logaritmo discreto. Cominciamo con l'osservare che se $x = \log_g y$, allora è sufficiente conoscere x modulo p^a per ogni $p \mid n$ con a il massimo esponente di p nella fattorizzazione di n. Infatti, scritto $n = p_1^{a_1} \cdot \ldots \cdot p_s^{a_s}$, se conoscessimo

$$x_1 = x \pmod{p_1^{a_1}}, x_2 = x \pmod{p_2^{a_2}}, \dots, x_s = x \pmod{p_s^{a_s}},$$

possiamo ricavare x tramite il teorema cinese dei resti.

Anche in questo algoritmo vi è un passaggio di precomputazione, dipendente solo dal gruppo G su cui si opera, e un passo che, dato $y \in G$, cerca $x = \log_g y$. Procediamo con la descrizione dei passi:

1. Si fattorizza $n = |G|$ ottenendo
$$n = \prod_{i=1}^{s} p_i^{a_i}.$$
Per ogni $p \mid n$ si calcolano le radici p-esime dell'unità:
$$r_{j,p} = g^{\frac{n}{p} j} \qquad \text{con } j = 0, \dots, p-1.$$

2. Per ogni $p \mid n$, detto a il massimo esponente di p nella fattorizzazione di n, cerchiamo $x \pmod{p^a}$. Scriviamo x in base p:
$$x \pmod{p^a} = x_0 + x_1 p + x_2 p^2 + \dots x_{a-1} p^{a-1} \qquad \text{con } 0 \leqslant x_j \leqslant p-1.$$
L'obiettivo è ricavare i $x_1, \dots, x_{a-1}$. Descriviamo la ricerca di x_0, come vedremo il procedimento si generalizza per ogni x_j una volta che si conoscono i precedenti. Calcoliamo $y^{\frac{n}{p}}$, dove
$$\begin{aligned} y^{\frac{n}{p}} &= g^{x \frac{n}{p}} = g^{\frac{n}{p}(x_0 + x_1 p + \dots + x_{a-1} p^{a-1})} = g^{x_0 \frac{n}{p} + (x_1 + x_2 p + \dots + x_{a-1} p^{a-1}) n} \\ &= g^{x_0 \frac{n}{p}} g^{t n} = g^{x_0 \frac{n}{p}}, \end{aligned}$$
d'altronde, per costruzione, $0 \leqslant x_0 \leqslant p-1$, per cui
$$g^{x_0 \frac{n}{p}} = r_{j_0,p} \implies x_0 = j_0$$
per qualche j_0 trovato nel passo precedente. Per la ricerca di x_1 si applica un procedimento analogo calcolando
$$y_1^{\frac{n}{p^2}} \qquad \text{dove } y_1 = y\, g^{-x_0}.$$
Ora
$$y_1^{\frac{n}{p^2}} = g^{\frac{n}{p^2}(x_1 p + x_2 p^2 + \dots + x_{a-1} p^{a-1})} = g^{x_1 \frac{n}{p}} g^{t_1 n} = g^{x_1 \frac{n}{p}}.$$
Nuovamente $0 \leqslant x_1 \leqslant p-1$ implica che esiste j_1 tale che
$$g^{x_1 \frac{n}{p}} = r_{j_1,p} \implies x_1 = j_1.$$
Il procedimento, iterato per gli altri x_i, ci fornisce
$$x \pmod{p^a} = j_0 + j_1 p + j_2 p^2 + \dots + j_{a-1} p^{a-1}.$$

3. Il passo precedente, iterato per tutti gli p^a della fattorizzazione di n ci fornisce
$$x \pmod{p_1^{a_1}}, x \pmod{p_2^{a_2}}, \dots, x \pmod{p_s^{a_s}},$$
tramite i quali possiamo calcolare x applicando il teorema cinese dei resti.

Osservazione. Questo algoritmo è efficiente quando l'ordine del gruppo G non ha fattori grandi.

Esempio. Consideriamo il gruppo $G = \mathbb{Z}_{37}^*$, il cui ordine è $n = 36$, e il suo generatore $g = 2$. Svolgiamo il primo passo di precomputazione:

1. $n = 36 = 2^2 3^2$ quindi dobbiamo calcolare le radici 2-esime e 3-esime dell'unità:

- Per $p = 2$ avremo
$$r_{0,2} = 1 \qquad \text{e} \qquad r_{1,2} = -1.$$
- Per $p = 3$ avremo
$$r_{0,3} = 1 \qquad r_{1,3} = 26 \qquad r_{2,3} = 10$$

2. Supponiamo di voler calcolare il logaritmo discreto x di $y = 28$. Dobbiamo trovare
$$x \pmod{2^2} \qquad \text{e} \qquad x \pmod{2^3}.$$
- Per $p = 2$ scrivo $x \pmod{2^2} = x_0 + x_1 \cdot 2$. x_0 si ottiene calcolando
$$y^{\frac{n}{2}} = 1 = r_{0,2} \implies x_0 = 0.$$
x_1 si ottiene da $y_1 = y\, g^{-x_0}$ calcolando
$$y_1^{\frac{n}{2^2}} = -1 = r_{1,2} \implies x_1 = 1.$$
Pertanto $x \pmod{2^2} = 0 + 1 \cdot 2 = 2$.
- Per $p = 3$ scrivo $x \pmod{3^2} = x_0 + x_1 \cdot 3$. x_0 si ottiene calcolando
$$y^{\frac{n}{3}} = 26 = r_{1,3} \implies x_0 = 1.$$
x_1 si ottiene da $y_1 = y\, g^{-x_0}$ calcolando
$$y_1^{\frac{n}{3^2}} = 10 = r_{2,3} \implies x_1 = 2.$$
Pertanto $x \pmod{3^2} = 1 + 2 \cdot 3 = 7$

3. Impostando il sistema
$$\begin{cases} x \equiv 2 \pmod 4 \\ x \equiv 7 \pmod 9 \end{cases} \implies \begin{cases} x \equiv -2 \pmod 4 \\ x \equiv -2 \pmod 9 \end{cases}$$
si ottiene che $x = \log_g y = -2 = 36$.

4.7.3 ALGORITMO INDEX-CALCULUS

Questo algoritmo, l'ultimo che studieremo per la ricerca del logaritmo discreto, è un algoritmo speciale per i gruppi $\mathbb{F}_q^*$, noi vedremo in particolare il caso per $q = p$, ovvero per $\mathbb{F}_q^* = \mathbb{Z}_p^*$.

Come per alcuni algoritmi di fattorizzazione studiati in precedenza, sfrutteremo i numeri B-smooth per qualche insieme di primi B. Consideriamo quindi un intero b che funge da limite superiore e prendiamo un insieme B si b primi piccoli:
$$B = p_1, p_2, \dots p_b \qquad \text{con } p_i \text{ primi piccoli distinti.}$$

Procediamo con la descrizione dell'algoritmo:

1. Supponiamo di trovare $c \geqslant b$ interi distinti $x_1, \dots, x_c$ tali che
$$g_i = g^{x_i} \pmod p$$
sia un B-numero. Quindi
$$g_i = \prod_{j=1}^{b} p_i^{a_{ij}} \pmod p \qquad \text{con } a_{ij} \geqslant 0.$$
In ognuna di queste c uguaglianze posso passare ai logaritmi
$$g_i = g^{x_i} = \prod_{j=1}^{b} p_j^{a_{ij}} \implies x_i = \sum_{j=1}^{b} a_{ij} \log_g p_j \pmod{p-1}.$$

Quindi otteniamo il seguente sistema lineare

$$\begin{cases} x_1 = \sum_{j=1}^{b} a_{1j} \log_g p_j \pmod{p-1} \\ \vdots \\ x_c = \sum_{j=1}^{b} a_{cj} \log_g p_j \pmod{p-1} \end{cases}$$

Di tali equazioni conosciamo gli x_i e a_{ij}. Abbiamo quindi un sistema lineare nelle incognite $\log_g p_i$ con $i = 1, \dots, b$. Dal momento che vi sono b incognite e c equazioni con $b \leqslant c$ avremo certamente una soluzione. Risolvendo il sistema otteniamo quindi

$$\log_g p_1, \log_g p_2, \dots, \log_g p_b,$$

per ogni $p_i \in B$.

2. Dato y vogliamo trovare x tale che $g^x = y$. Troviamo x^*, eventualmente anche $x^* = 0$ tale che

$$g^{x^*} y$$

sia B-smooth. In tal caso avremo

$$g^{x^*} y = \prod_{i=1}^{b} p_i^{a_i^*} \implies x^* + \log_g y = \sum_{i=1}^{b} a_i^* \log_g p_i,$$

da cui

$$x = \log_g y = -x^* + \sum_{i^1}^{b} a_i^* \log_g p_i.$$

Osservazione. La complessità di questo algoritmo è stimata essere sottoesponenziale. Inoltre la sua efficienza non dipende dalla scelta del primo p.

Esempio. Sia $p = 101$, lavoriamo quindi sul gruppo $G = \mathbb{Z}_{101}^*$ con il generatore $g = 3$. Cominciamo con il passo di precomputazione:

1. Scegliamo casualmente alcuni x_i, calcoliamo $g_i = g^{x_i}$ e troviamone la loro fattorizzazione. Il nostro obiettivo è trovare più x_i del numero di primi necessari a rendere i g_i B-smooth, dove B è il nostro insieme di primi.
   - Prendiamo $x_1 = 10$, avremo
   $$g_1 = g^{x_1} = 65 \pmod{p} = 5 \cdot 13,$$
   per cui $5, 13 \in B$.
   - Prendiamo $x_2 = 12$, avremo
   $$g_2 = g^{x_2} = 80 \pmod{p} = 2^4 \cdot 5,$$
   per cui $2 \in B$.
   - Prendiamo $x_3 = 14$, avremo
   $$g_3 = g^{x_3} = 13 \pmod{p}.$$

   Abbiamo quindi trovato tre elementi tali che i rispettivi g_i sono B-smooth su
   $$B = \{2, 5, 13\},$$
   che ha a sua volta tre elementi. Scriviamo quindi il sistema lineare passando ai logaritmi
   $$\begin{cases} x_1 = \log_g 5 + \log_g 13 \pmod{p-1} \\ x_2 = 4 \log_g 2 + \log_g 5 \pmod{p-1} \\ x_3 = \log_g 13 \pmod{p-1} \end{cases} \implies \begin{cases} \log_3 13 = 14 \pmod{100} \\ \log_3 5 = 96 \pmod{100} \\ 4 \log_3 2 = 16 \pmod{100} \end{cases}$$

Osserviamo che, dal momento che $(4, 100) \mid 16$, la congruenza $4 \log_3 2 = 16 \pmod{100}$ ha precisamente $(4, 100) = 4$ soluzioni modulo 100, che sono $4, 29, 54, 79$. Per trovare la soluzione che effettivamente risolve il logaritmo discreto di 2 si procede per tentativi, nel nostro caso troviamo

$$\log_3 2 = 29.$$

2. Supponiamo di voler trovare il logaritmo discreto di $y = 87$. Osserviamo che, preso $x^* = 5$, si ha

$$g^{x^*} y = 3^5 \cdot 87 = 32 \pmod{p} \qquad \text{dove } 32 = 2^5,$$

per cui $g^{x^*} y$ è B-smooth. Otteniamo quindi la soluzione passando al logaritmo:

$$x^* + \log_g y = 5 \log_g 2 \iff 5 + \log_3 87 = 5 \cdot 29 \implies \log_3 87 = 40.$$

5 FIRMA DIGITALE

La firma è la componente del documento che certifica la provenienza del messaggio. Per essere valida deve far parte fisicamente del documento, quindi non apposta su un supporto di quest'ultimo, dovrebbe essere verificabile e difficile da falsificare. Anche nel caso di uno scambio di messaggi all'interno di un crittosistema è importante essere certi di chi sia il mittente. In un crittosistema simmetrico, la conoscenza della chiave da entrambe le parti rende superflua una firma aggiuntiva in quanto una volta che il messaggio decifrato ha senso, la legittima provenienza è garantita. D'altronde, in un crittosistema a chiave pubblica, chiunque può scrivere un messaggio cifrato a Bob affermando di essere Alice. Pertanto è necessaria una firma digitale.

Questa firma dovrà avere le stesse caratteristiche di una firma fisica, in particolare

- Deve verificare l'identità del mittente.
- Deve essere vincolante a livello legale in modo che il mittente non possa ripudiare ciò che ha firmato.
- Deve garantire che il messaggio non sia stato alterato dopo l'invio.

5.1 SCHEMI DI FIRMA

Uno schema di firma si basa tipicamente su crittosistemi a chiave pubblica e si compone di due algoritmi:

- Un algoritmo che produce la firma sulla base del messaggio e di una chiave privata.
- Un algoritmo di verifica che si basa sul messaggio ricevuto e su una chiave pubblica del mittente.

Quindi per firmare il messaggio x, Alice usa l'algoritmo s_k che dipende da una chiave k, e calcola $y = s_k(x)$. Viceversa, data una coppia (x, y), dove x è il messaggio e y la firma, l'algoritmo $v_k(x, y)$ ritorna vero se y è una firma valida di x e falso altrimenti.

Osservazione. In questo momento, e nella descrizione dei prossimi schemi di firma, non si richiede che (x, y) sia cifrato. Accenneremo alle implementazioni reali che tengono conto della cifratura.

Definizione 5.1 – Schema di firma

Uno *schema di firma* è una 5-pla (P, A, K, S, V) dove

- P è un insieme finito di possibili messaggi.
- A è un insieme finito di possibili firme.
- K, lo spazio delle chiavi, è un insieme finito di possibili chiavi.
- Per ogni $k \in K$ c'è un algoritmo si firma $s_k \in S$ e un corrispondente algoritmo di verifica $v_k \in V$.
- Le mappe $s_k \colon P \to A$ e $v_k \colon P \times A \to \{\text{Vero}, \text{Falso}\}$ sono tali che, per ogni $x \in P$ e per ogni $y \in A$, vale

$$v_k(x,y) = \begin{cases} \text{Vero} & \text{se } y = s_k(x) \\ \text{Falso} & \text{se } y \neq s_k(x) \end{cases}$$

L'idea per uno schema di firma è, spesso, quello di usare un crittosistema a chiave pubblica. Tipicamente il mittente è in possesso di una propria chiave k_A, composta da una parte e_{k_A} pubblica ed una d_{k_A} privata. Si potrebbe quindi firmare il messaggio x ponendo

$$s_{k_A}(x) = y = d_{k_A}(x),$$

che renderebbe privato l'algoritmo di firma in quanto il mittente è l'unico a poter utilizzare d_{k_A} per decifrare. A questo punto si invierebbe la coppia (x, y) e, per verificare la firma, si calcolerebbe

$$e_{k_A}(y).$$

Se $e_{k_A}(y) = x$ allora l'algoritmo di verifica riporterebbe esito positivo.

Osservazione. Nelle implementazioni reali, a questo procedimento segue la cifratura della coppia (x, y). Osserviamo che è importante cifrare dopo aver firmato, poiché, scambiando tale ordine, è possibile convincere il destinatario di essere il mittente di un messaggio originariamente inviato da qualcun altro. Infatti, supponiamo che x sia il messaggio. Alice cifra $z = e_{k_B}(x)$, firma $y = s_{k_A}(z)$ e invia la coppia (z, y) a Bob. Se Eve intercetta la trasmissione è in grado di firmare il messaggio z, pur non potendo decifrarlo. Può infatti calcolare $\tilde{y} = s_{k_E}(z)$ e inviare la coppia $(z, \tilde{y})$ a Bob. Tale messaggio passerà la verifica di Bob che lo accetterà come proveniente da Eve.

Proviamo ad applicare quanto detto al crittosistema RSA. Sia $N = p\,q$ con p, q primi. Sia $P = A = \mathbb{Z}_N$, lo spazio delle chiavi è

$$K = \{\, (N, p, q, d, e) \mid e\,d \equiv 1 \pmod{\varphi(N)} \,\}.$$

(N, e) è la chiave pubblica, (p, q, d) la chiave privata. Se $k = (N, p, q, d, e)$ è una chiave, si pone

$$s_k(x) = x^d \pmod N \qquad \text{e} \qquad v_k(x,y) = \begin{cases} \text{Vero} & \text{se } x \equiv y^e \pmod N \\ \text{Falso} & \text{altrimenti} \end{cases}$$

Esempio. La chiave RSA di Alice è

$$k_A = (N_A, p_A, q_A, d_A, e_A) = (2773, 47, 59, 17, 157).$$

Alice vuole firmare e trasmettere il messaggio $x = 920$. Utilizza quindi la chiave privata $d_A = 17$ e calcola

$$s_{k_A}(x) = 920^{17} \equiv 948 \pmod N.$$

La coppia (messaggio, firma) è pertanto $(920, 948)$. Bob riceve

$$(x, y) = (920, 948),$$

per verificare la firma y utilizza la chiave pubblica di Alice $e_A = 157$ e calcola

$$y^{e_A} = 948^{157} \equiv 920 = x \pmod{N},$$

per cui $v_{k_A}(x, y)$ ritorna vero.

Osservazione. Per combinare firma e cifratura, Alice genera la coppia (messaggio firma) (x, y) con il metodo visto prima. A questo punto cifra la coppia come se fosse il messaggio da inviare a Bob, pertanto, utilizzando la chiave pubblica di Bob, calcola

$$z = e_{k_B}((x, y)).$$

Bob riceve z e decifrandolo con la sua chiave privata ottiene (x, y). A questo punto utilizza l'algoritmo di verifica per controllare la firma.

5.2 SCHEMA DI ELGAMAL

Lo schema di firma basato sul crittosistema di Elgamal è lo schema standard utilizzato negli Stati Uniti. Osserviamo che, non trattandosi di un crittosistema deterministico, lo schema di firma non sarà lineare come nel caso di quello applicato ad RSA. Consideriamo la chiave

$$k_A = (p, g, a, \beta) \qquad \text{con } \beta = g^a \pmod{p},$$

dove a è privato. Supponiamo che Alice debba firmare $x \in \mathbb{Z}_p^*$. Allora sceglie casualmente $h \in \{2, \dots, p-2\}$ tale che $(h, p-1) = 1$, in modo tale che esista l'inverso di h modulo $p-1$. Avremo

$$s_{k_A}(x, h) = (z_1, z_2) \qquad \text{dove } \begin{cases} z_1 = g^h \pmod{p} \\ z_2 = (x - a\, z_1)\, h^{-1} \pmod{p-1} \end{cases}$$

La verifica viene eseguita da Bob, il quale conosce g, p, β e deve verificare $\big(x, (z_1, z_2)\big)$. Calcola quindi

$$g^x \pmod{p} \qquad \text{e} \qquad \beta^{z_1} z_1^{z_2} \pmod{p},$$

e accetta il messaggio se

$$g^x \equiv \beta^{z_1} z_1^{z_2} \pmod{p}.$$

Infatti

$$\beta^{z_1} z_1^{z_2} = g^{a\, z_1} g^{h\,(x - a\, z_1)\, h^{-1}},$$

dove $h\, h^{-1} \equiv 1 \pmod{p-1}$, ovvero $g^{h\, h^{-1}} \equiv g \pmod{p}$. Da cui

$$\beta^{z_1} z_1^{z_2} \equiv g^{a\, z_1} g^{x - a\, z_1} = g^x \pmod{p}.$$

Esempio. Prendiamo $p = 107, g = 2, a = 11$, da cui $\beta = g^a \equiv 15 \pmod{p}$, la chiave è pertanto

$$(p, g, a, \beta) = (107, 2, 11, 15).$$

Supponiamo che Alice debba firmare $x = 10$. Sceglie casualmente $h = 7$ dove $(7, p - 1) = 1$ e calcola $7^{-1} \equiv 91 \pmod{p-1}$. Quindi calcola

$$z_1 = g^h \equiv 21 \pmod{p} \qquad \text{e} \qquad z_2 = (x - a\, z_1)\, h^{-1} \equiv 29 \pmod{p-1}.$$

Dopodiché trasmette a Bob la terna $\big(x, (z_1, z_2)\big) = \big(10, (21, 29)\big)$. Bob riceve la terna, eventualmente da decifrare, e per verificare calcola

$$g^x \equiv 61 \pmod{p} \qquad \text{e} \qquad \beta^{z_1} z_1^{z_2} \equiv 61 \pmod{p}.$$

Dal momento che le quantità coincidono modulo p, accetta il messaggio come autentico.

5.3 ATTACCHI AD UNO SCHEMA DI FIRMA

In un attacco ad uno schema di firma, si vuole falsificare una firma in modo che passi il test di verifica come se fosse stato firmato dal mittente originale. In generale, non è possibile firmare un messaggio arbitrario x, d'altronde è facile produrre una coppia (x, y) che passi il test nel momento in cui non si chieda alcuna condizione su x. Ad esempio, per lo schema di firma RSA, la coppia

$$(e_A(y), y),$$

passa la verifica per ogni y.

Notazione. La falsificazione appena descritta si chiama *falsificazione esistenziale*, poiché si produce una coppia che supera la verifica ma senza avere alcun controllo sul messaggio.

Nelle implementazioni reali, questo tipo di attacco non è praticabile in quanto non si firma mai il messaggio x bensì un *hash* di x, ovvero una stringa ottenuta in modo unidirezionale da x.

INDICE ANALITICO

Algoritmo esponenziale, 9
Algoritmo polinomiale, 9

B-smooth, 45
B-vettore, 45

Crittosistema, 4

Grandezza di un intero, 8

Logaritmo discreto, 48

Numero
    di Carmichael, 21
    di Fermat, 19

O grande, 7

Operazioni bit, 7

Pseudoprimo, 21
    di Eulero, 26
    forte, 29

Radice primitiva, 22
Residuo quadratico, 22

Schema di firma, 60
Simbolo
    di Jacobi, 24
    di Legendre, 23

Test
    deterministico, 20
    probabilistico, 20