Numeri primi e problema della primalità

Fino all'inizio del secolo scorso, i due problemi: stabilire se un numero intero n è primo, e, nel caso non lo sia, fattorizzarlo in prodotto di primi, sono stati considerati equivalenti.

L'idea classica era quella di provare a dividere un numero n dispari per tutti i suoi predecessori (dispari):

Se non si trova un divisore allora n è primo, altrimenti si è trovato un fattore proprio.

Il meglio che si può fare con questa strategia è fermarsi alla radice quadrata di n, ma per numeri molto grandi si tratta di un procedimento del tutto ineffi-ciente.

L'idea che si possa stabilire che un numero è composto senza fattorizzarlo inizia con il **teorema di Fermat**:

Se p è primo e a è primo con p, si ha che

$$a^{p-1} = 1 \, mod \, p$$

Putroppo questa condizione è solo necessaria, e quindi può servire solo a stabilire che n non è primo: se infatti si trova un a che non soddisfa l'enunciato del teorema, certamente n è composto.

Notare che non si fa alcun riferimento alla fattorizzazione, e non si ha alcuna informazione sui fattori primi di n.

Non è possibile utilizzare il teorema di Fermat per stabilire che un numero è primo: esistono infatti numeri composti che vericano la condizione del teorema, per ogni a (base ammissibile).

Si chiamano *Numeri di Carmichael.*

Non si sa molto su questi numeri, ma sono infiniti, e relativamente frequenti.

I due problemi, **primalità** e **fattorizzazione**, si separano attorno agli anni '60 del secolo scorso, con il test di primalità probabilistico di Miller-Rabin.

Miller raffina la condizione di Fermat, in modo che diventi un criterio non solo **necessario**, ma anche **sufficiente**, per stabilire che un numero è primo.

Test di Miller

Sia n un intero positivo dispari. Diciamo che **n passa il test di Miller** rispetto a una base ammissibile a, se posto

$$n-1=2^s t$$

con t dispari, si verifica una delle due seguenti condizioni (mutuamente esclusive):

$$a^t = 1 \, mod \, n$$

oppure esiste un r < s tale che

$$a^r = -1 \, mod \, n$$

Abbiamo allora che **n è primo se e solo se passa il test di Miller rispetto a tutte le basi ammissibili a**.

Queste però sono ancora troppe: sono φ(n), dove φ è la funzione di Eulero, e φ(n) è dell'ordine di n. Questo rende il procedimento del tutto inefficiente.

Il problema viene risolto mediante il

Teorema di Rabin

Sia n composto dispari. Allora le basi ammissibili per cui n passa il test di Miller sono al più $\varphi(n)/4$.

Questo permette di utilizzare il risultato di Miller in modo probabilistico, ottenedo così il

Test di primalità probabilistico di Miller – Rabin

Sia n un intero dispari, a una base ammissibile per n (MCD(a, n) = 1).

Applichiamo il test di Miller ad n, a.

Se il test **non** viene superato, n è **certamente** composto.
Se il test è superato, n è **primo**, con una probabilità di errore minore di 1/4.

Ripetiamo il procedimento con k basi ammissibili scelte in modo indipendente. Se otteniamo sempre esito positivo, possiamo concludere che n è primo, con probabilità di errore che tende a 0 come $1/4^k$, quindi con velocità esponenziale.

Attualmente il test di Miller-Rabin è usato universalmente nella generazione di numeri primi, ad esempio per le chiavi di RSA.

Il numero di operazioni aritmetiche richieste dal test è polinomiale nel log n, quindi abbiamo un test effi-ciente.

Fattorizzazione

Per quanto riguarda la fattorizzazione la situazione è invece fondamentalmente diversa.

- Non conosciamo algoritmi polinomiali (a parte il caso quantistico, per il quale esiste un algoritmo dovuto a Shor)

- Non sappiamo se questi algoritmi esistano o no.

Tutti gli algoritmi conosciuti attualmente sono almeno sub-esponenziali, uno dei più sofisticati è il **Crivello dei campi di numeri** (*Number Field Sieve*) di Pollard (1988), che ha un tempo di esecuzione

$$\left(\mathrm{e}^{(c+o(1))n^{1/3}(\log n)^{2/3}}\right)$$

che è ancora lontano dall'essere polinomiale.

La maggior parte dei metodi moderni di fattorizzazione si fonda ancora su una strategia del tutto elementare:

Trovare x, y tali che

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

che risale almeno a Fermat.

Esempio: metodo di fattorizzazione di Fermat

Fermat cerca di rappresentare n come differenza di due quadrati. La sua strategia consiste nel cercare un x tale che $x^2 - n$ sia un quadrato perfetto. Per limitare lo spazio della ricerca inizia con $x = [\sqrt{n}] + 1$ e procede incrementando via via di una unità, fino a quando non trova un quadrato perfetto.

Notare che si può sempre rappresentare n come differenza di due quadrati, infatti se n = ab basta porre

$$x = \frac{a+b}{2} \qquad\qquad y = \frac{a-b}{2}$$

il problema nasce nel caso in cui n è primo (Fermat non aveva un test di primalità), in cui a = n e b = 1. In questo caso, per scoprire che n è primo, e quindi i fattori sono banali, occorre arrivare fino a $x = \frac{a+b}{2}$

In questo caso il metodo è addirittura peggiore delle divisioni per tentativi, nel quale basta arrivare solo fino a $\sqrt{n}$.

Il metodo ha qualche possibilità di successo se n è “quasi” un quadrato, ossia se è il prodotto di due fattori molto vicini.

Esempio: metodo ρ (rho) di Pollard

E' uno dei pochi che non usa la relazione precedente. Consiste essenzialmente nell'eseguire una “passeggiata a caso” nell'anello degli interi mod n, partendo da un punto iniziale scelto in modo casuale, e applicando in modo iterativo una funzione “casuale” F:

$$x_0 \in Z_n$$

$$x_1 = F(x_0)$$
$$x_2 = F(x_1)$$

$$\vdots$$

$$x_{i+1} = F(x_i)$$

Dato che Z_n è finito, dopo un numero finito di passi si troveranno x_j , x_k , con j < k, tali che $x_j = x_k \, mod \, n$.
La speranza, in realtà, è che, se r è un fattore prorprio di n, si trovino prima due elementi tali che

$$x_j = x_k \, mod \, r$$

ma non

$$x_j = x_k \, mod \, n$$

In questo caso $MCD(x_j - x_k, n)$ dà proprio r.

Il problema è che non conosciamo r (lo stiamo cercando !), e siamo costretti a eseguire i test sul MCD a ogni passo, per tutte le possibili differenze, rendendo l'algoritmo piuttosto inefficiente.

Pollard ha escogitato diverse ottimizzazioni molto ingegnose, che permettono di ridurre il numero dei MCD da calcolare a ogni passo a uno solo.

Esempio

n = 4087, $x_0=2$, $F(x)=x^2+x+1$

$x_0=2$

$x_1=7$ $\qquad MCD(x_1-x_0, n)=1$

$x_2=57$ $\qquad MCD(x_2-x_1, n)=1$

$x_3=3307$ $\qquad MCD(x_3-x_1, n)=1$

$x_4=2745$ $\qquad MCD(x_4-x_3, n)=1$

$x_5=1343$ $\qquad MCD(x_5-x_3, n)=1$

$x_6=2626$ $\qquad MCD(x_6-x_3, n)=1$

$x_7=3734$ $\qquad MCD(x_7-x_3, n)=61$

Abbiamo così scoperto che 61 è un fattore proprio di 4087, calcolando ad ogni passo un solo MCD.

Nonostante le ottimizzazioni, l'efficienza dell'algoritmo rimane limitata. Si può dimostrare che:

Se F, x_0 sono scelti in modo uniformemente casuale, l'algoritmo trova un fattore r con probabilità elevata, e con un numero di operazioni binarie

$$O(\sqrt[4]{n}\log^3(n))$$

La probabilità di insuccesso può essere resa minore di e^{-k} aumentando il numero di operazioni di un fattore $\sqrt{k}$

Esempio: Prendiamo k = 9**,** in modo che la probabilità di errore sia minore di $e^{-9} \simeq 10^{-3}$

Il numero di operazioni binarie richieste è dell'ordine di

$3\sqrt[4]{n}\log^3(n)$. Nelle applicazioni crittografiche (ad esempio RSA) è normale un valore di n dell'ordine di 10^{300} , e quindi il metodo richiede circa 10^{81} operazioni, il che significa 3×10^{64} anni, su un processore che esegue un'operazione binaria in 10^{-9} secondi.

Il sistema di crittografia a chiave pubblica RSA (il primo e tuttora il più utilizzato) fonda la sua sicurezza sulla (supposta) difficoltà della fattorizzazione :

Fattorizzando il modulo (che è parte della chiave pubblica) è possibile ricavare la chiave privata.

Di solito questo non è considerato un problema, perché la fattorizzazione è considerata comunque intrattabile. Ma in realtà la difficoltà non è sempre quella del caso peggiore: esistono casi in cui RSA può essere forzato con metodi matematici elementari, ad esempio con il *Teorema Cinese dei Resti*, se i parametri non sono scelti in modo corretto.

Teorema cinese dei resti

Siano $n_{1,} \cdots n_k$ interi positivi due a due coprimi, $a_{1,} \cdots a_k$ interi qualsiasi. Il sistema di congruenze:

$$x = a_1 \, mod \, n_{1,} \cdots, \, x = a_k \, mod \, n_k$$

ha una soluzione *unica* mod M, con $M = a_1 \cdots a_k$.

Ciò significa che esiste un unico intero x compreso tra 0 e m-1 che soddisfa tutte le congruenze, ciascuna rispetto al suo modulo. Come si trova x ?

Poniamo $N_i = M / n_i$ e $M_i = N_i^{-1} \, mod \, n_i$. Allora:

$$x = \sum_{i=1}^{k} a_i M_i N_i .$$

Esempio: attacco broadcast di Hastad a RSA con e piccolo.

Supponiamo che Bob voglia mandare uno stesso messaggio M a k destinatari diversi, ognuno con una sua chiave pubblica (e_i, n_i), e supponiamo che gli esponenti di cifratura siano tutti uguali a e (i moduli rimangono diversi).

L'avversario Marvin intercetta $C_{1,} \cdots C_k$, e vale

$$C_1 = M^e \bmod n_{1,} \cdots, C_k = M^e \bmod n_k$$

Se i moduli sono scelti in modo casuale, è probabile che siano anche due a due coprimi. Con il Teorema cinese dei resti calcola C tale che

$C = M^e \bmod n_{1,} \cdots, C = M^e \bmod n_k$ e quindi

$$C = M^e \bmod n_1 \ldots n_k$$

Se e < k, allora da $M < n_{1,} \cdots, M < n_k$ segue che $M^e < n_{1,} \cdots n_k$, e l'eguaglianza $C = M^e$ vale sugli interi. A questo punto è possibile ricavare M estraendo (numericamente) la radice e-sima di C sui numeri reali.

Naturalmente l'attacco funziona solo se l'esponente e è abbastanza piccolo.

Esempio: fattorizzazione del modulo N di RSA conoscendo la chiave privata d (con il TCR).

Sia N = pq, con p, q primi. Segue dal TCR che in Z_N esistono esattamente 4 radici quadrate dell'unità. Due sono ± 1 , le altre due sono della forma $\pm x$, con $x \neq 1 \; mod \; N$. Supponiamo di conoscere entrambe le chiavi di RSA:

e = chiave pubblica

d = chiave privata

Allora possiamo calcolare k = ed – 1. Per il modo in cui sono definite le chiavi, k è un multiplo di φ(n). Dal Teorema di Eulero segue che per ogni $g \in Z_N - (0)$ si ha $g^k = 1 \; mod \; N$, e quindi $x = g^{k/2}$ è una radice quadrata di 1 mod N. Conoscere una radice quadrata dell'unità diversa da 1 e -1 permette di fattorizzare N : MCD (x – 1, N) dà uno dei fattori primi di N.

Return Of the Coppersmith Attack

Nel 1997 Don Coppermith ha inventato un algoritmo in grado di calcolare trovare tutte le soluzioni di una equazione modulo N con coefficienti interi

$$f(x) = 0 \, mod \, N$$

purché queste siano abbastanza “piccole”, ossia soddisfino

$$x < N^{1/d}$$

dove d è il grado del polinomio f(x), e in tempo polinomiale in $(logN, 2^d)$.

L'algoritmo si basa sul fatto che le radici piccole modulo N di f(x) sono anche radici intere di un opportuno polinomio g, e quindi possono essere trovate facilmente mediante approssimazione numerica.

Per trovare g si costruisce una matrice a coefficienti interi a partire dai coefficienti di f, si considera il reticolo discreto generato dalle righe di questa matrice, e si applica l'algoritmo LLL (Lenstra, Lenstra e Lovàsz) che permette di trovare un vettore non nullo di norma piccola in un reticolo.

Come conseguenza si ha un risultato, abbastanza sorprendente, sulla sicurezza di RSA:

Sia N = PQ, con P, Q primi distinti. E' possibile trovare P e Q in tempo polinomiale conoscendo solo

$$(1/4)\log_2 N$$

dei bit di P, i più significativi, o anche i meno significativi. Quindi conoscendo solo ¼ dei bit di uno dei fattori, è possibile fattorizzare N, e quindi ricavare la chiave segreta.

L'idea è semplice: l'algoritmo funziona (opportunamente adattato) anche in due variabili. Supponiamo sia P = p + x, Q = q + y, con p, q le parti note di P e Q. Allora N = (p + x)(q + y) = N + py + qx + xy, e possiamo trovare x, y come radici del polinomio f(x, y) = xy +qx + py mod N, se queste sono abbastanza piccole.

Trovare anche solo ¼ dei bit di un fattore primo di N non è facile, in generale, se i primi P, Q sono generati in modo corretto (con entropia sufficientemente grande), e quindi questa tecnica non è stata considerata un pericolo reale per RSA, solo un'indicazione del fatto che i fattori primi di N devone essere protetti adeguatamente.

Nell'ottobre del 2017 un gruppo di ricercatori ha scoperto un fatto interessante: in una libreria crittografica incorporata in moltissime smartcard e altri dispositivi hardware, l'implementazione di RSA è vulnerabile ad una variazione dell'attacco di Coppersmith (20 anni dopo !). Questo perché i primi P, Q usati nel modulo N = PQ sono generati in modo non sufficientemente casuale: sono tutti della forma

$$P = k \times M + \left(65537^{a} \, mod \, M\right)$$

L'intero M è noto, è il prodotto di un certo numero n di primi successivi:

$$M = 2 \times 3 \times 5 \times .... P_{n}$$

con n che dipende dalla lunghezza della chiave, e può valere 71, 126, 225.
$65537 = 2^{16} + 1$ è un primo, ed è l'esponente di cifratura usato invariabilmente da *tutti* I siti web nel protocollo SSL (o TLS). Solo k e a non sono conosciuti. M è grande, quasi dello stesso ordine di grandezza di P, di conseguenza k e a sono piccoli: in una implementazione di RSA a 512 bit, con primi a 256 bit, k ha 37 bit e a ne ha 62, e P, di fatto, ne ha solo 99.
Per trovare k si cerca una radice piccola del polinomio (con qualche semplificazione)

$$f\left(x\right) = x \times M + \left(65537^{a} \, mod \, M\right)$$

provando diversi valori di a. Per un modulo a 1024 bit, il tempo stimato è di 31 giorni su Amazon c4, con un costo di 76 dollari (!).