una possibile funzione unidirezionale

- moltiplicare due interi a n bit è facile (in $\mathcal{O}(n^2)$ con l'algoritmo usuale)
- trovare un primo a n bit, e verificare che è primo, è facile
- fattorizzare un numero a n bit è difficile ($2^{cn^{1/3}}$)
- si può costruire un crittosistema a chiave pubblica basato su questa osservazione?

crittosistema RSA

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

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

- Se $k = (N, p, q, d, e)$ è una chiave, poniamo
- $e_k(x) = x^e \pmod N$
- N e e sono la chiave pubblica
- $d_k(y) = y^d \pmod N$
- p, q, d sono la chiave privata

Problemi facili

1. dato un intero N, vedere se è primo
2. dati e e M numeri naturali, trovare (e, M); se è 1, calcolare l'inverso di e modulo M (alg. di Euclide – polinomiale)
3. calcolare la f.ne $x \to x^e \pmod N$

Problemi difficili

4. dato un intero N, fattorizzarlo
5. dato un intero N, calcolare $\phi(N)$
6. dati N e e, trovare d tale che $(x^e)^d = x \pmod N$

Problemi facili

1. dato un intero N, vedere se è primo (test di primalità)
2. dati c e M, trovare (c, M); se è 1, calcolare l'inverso di c modulo M (alg. di Euclide – polinomiale)
3. calcolare la f.ne $x \to x^e \pmod N$ (square and multiply)

problemi facili - calcolare $x \to x^e$

- sia $\mathbf{l}(e)$ la lunghezza di e scritto in base 2 ($\approx \log_2 e$) , sia n la lunghezza di N; $n = \mathbf{l}(N)$; ho $e < N$, quindi $\mathbf{l}(e) \leq n$
- $x^e = x \cdot x \cdot \ldots x$ (e fattori - quindi $\approx 2^{\mathbf{l}(e)}$ fattori - esponenziale)
- l'algoritmo square and multiply calcola $x \to x^e \pmod N$ usando al più $2\mathbf{l}(e)$ moltiplicazioni - e divisioni.
- scrivere e in base 2; $e = 2^{b_1} + 2^{b_2} \ldots 2^{b_k}$, con $b_1 + 1 = \mathbf{l}(e)$
- elevando al quadrato b_1 volte, calcolare $x^2, x^{2^2}, \ldots, x^{2^{b_1}} \pmod N$
- moltiplicando e riducendo $\pmod N$ si ha $x^e = x^{2^{b_1}} \cdot x^{2^{b_2}} \cdot x^{2^{b_k}}$
- con meno di $2\mathbf{l}(e)$ moltiplicazioni – quindi polinomiale

problemi "facili" - primalità

- ci sono infiniti numeri primi (Euclide)
- la dimostrazione è per assurdo (è la più celebre dimostrazione per assurdo): supponiamo ci siano solo un numero finito di primi, e siano questi $p_1, p_2, \ldots, p_k$
- consideriamo il numero $p_1 \cdot p_2 \cdot \cdots \cdot p_k + 1$
- questo numero non è divisibile per nessuno dei $p_1, p_2, \ldots p_k$ – e allora quali sono i suoi fattori primi?
- ci sono infiniti primi – quanti sono i numeri primi che precedono un dato intero x?

teorema dei numeri primi

- $\pi(x)$ = numero di primi p con $p \leq x$
- esempio: $\pi(30) = 4 + 4 + 2 = 10$
- teorema dei numeri primi: asintoticamente, $\pi(x) \approx \frac{x}{\log(x)}$
- congetturato da Legendre, Gauss fine 700 – dimostrato da Hadamard, de la Vallée Poussin fine 800
- dato un intero positivo N, la probabilità che un numero $< N$ scelto a caso sia primo è $\pi(N)/N$, per N molto grande $\approx 1/\log(N)$
- esempio: la probablità che un numero casuale con al più 100 cifre decimali sia primo è $\approx 1/\log(10^{100}) \approx 1/230$
- se sappiamo controllare se un intero è primo, trovare primi grandi è facile

test di primalità

- dato un intero N a n bit, come scoprire se è un primo?
- posso provare a fattorizzarlo per divisioni successive: devo fare al più $\sqrt{N}$ divisioni, quindi è in $\mathcal{O}(2^{n/2})$ – scopro se è primo, e trovo anche i fattori
- i test di primalità scoprono se N è primo senza dire niente dei suoi fattori
- fino al 2002 si conoscevano solo test di primalità polinomiali probabilistici, non deterministici
- nel 2002, M. Agrawal insieme a due suoi dottorandi, Kayal e Saxena, hanno trovato un algoritmo polinomiale per determinare la primalità
- inizialmente, in $\mathcal{O}(n^{12})$ – nel 2005, Lenstra e Pomerance lo portano a $\mathcal{O}(n^6)$
- primes è in P

test deterministici e test probabilistici

- un test deterministico dà una risposta certa: N è primo o non lo è
- un test probabilistico $\mathcal{T}$ consiste in una successione di test $\{\mathcal{T}_m\}_{m\in\mathbb{N}}$ e una successione che va a zero $\{\epsilon_m\}_{m\in\mathbb{N}}$ tale che,
  - se N non passa il test $\mathcal{T}_m$ allora non è primo,
  - la probabilità che N superi i test $\mathcal{T}_1, \ldots, \mathcal{T}_m$ e non sia primo è minore di ϵ_m
- i test di primalità probabilistici più usati sono quello di Solovay-Strassen (1977) e quello di Miller-Rabin (1980)
- il test di Miller-Rabin deriva da un test deterministico (dovuto a Miller) – che però assume l'ipotesi di Riemann generalizzata

idea

- per mostrare che un numero N non è primo si mostra che non si comporta come un primo
- si "cercano prove" del fatto che N non si comporta come un primo
- senza cercare i suoi fattori

PT di Fermat e test di Fermat

- il PTdF dice che, se p è un primo e $1 \leq a \leq p-1$, allora $a^{p-1} \equiv 1 \pmod{p}$
- vogliamo scoprire se N è primo – se trovo $a \leq N-1$ e $a^{N-1} \not\equiv 1 \pmod{N}$ sappiamo per certo che N non è primo – senza sapere niente sui suoi fattori
- questo è il test di Fermat
  - prendo $a < N$, calcolo (a, N); se $\neq 1$, N non è primo
  - se $(a, N) = 1$, calcolo a^{N-1} – se $\not\equiv 1 \pmod{N}$ allora N non è primo
- per esempio, $2^{322} \equiv 157 \pmod{323}$; quindi 323 non è primo $(323 = 17 \cdot 19)$

pseudoprimi

- se però $a^{N-1} \equiv 1 \pmod{N}$ – non posso concludere che N è primo
- per esempio $2^{340} \equiv 1 \pmod{341}$, ma $341 = 11 \cdot 31$
- si dice che N è uno pseudoprimo in base a se N non è primo ma $a^N \equiv a \pmod{N}$. 341 è uno pseudoprimo in base 2.
- possiamo provare diversi valori per a: per esempio $3^{341} \not\equiv 3 \pmod{341}$ – quindi 341 non passa il test!

numeri di Carmichael

- non basta: esistono numeri N che sono pseudoprimi in base a per ogni possibile base a dove $1 < a < N$ e $(a, N) = 1$
- un tale N si dice numero di Carmichael
- per esempio, si può vedere che 561 è un ndC (è il più piccolo)
- il test di Fermat non va bene
- dà comunque un idea di come può funzionare un test di primalità

Problemi difficili

1. dato un intero N, fattorizzarlo
2. dato un intero N, calcolare $\phi(N)$
3. dati N e e, trovare d tale che $(x^e)^d = x \pmod{N}$

- Se Eve riesce a fattorizzare $N = pq$, ottiene le info private di Bob: può calcolare sia $\phi(N)$ sia $d = e^{-1} \pmod{\phi(N)}$
- dunque violare l'RSA non può essere più difficile che fattorizzare

equivalenza fra i problemi difficili

- violare l'RSA non può essere più difficile che fattorizzare
- conoscere la fattorizzazione $\iff$ calcolare $\phi(N)$:
  - $\Rightarrow$ ovvio
  - $\Leftarrow$ conoscere N $(= pq)$ e $\phi(N)$ $(= (p-1)(q-1) = N - (p+q) + 1)$ vuol dire conoscere la somma $(N - \phi(N) + 1)$ e il prodotto (N) di p,q – quindi conoscere p e q
- i problemi 1 e 2 sono equivalenti: passo dall'uno all'altro in tempo polinomiale

l'esponente di decifratura

- supponiamo che Eve sappia risolvere il pb. 3: dati N e e, riesce a trovare d tale che $(x^e)^d = x \pmod N$
- allora può fattorizzare N
  - c'è un algoritmo polinomiale che fattorizza N oppure fallisce con probabilità minore di $1/2$ – ripetendolo più volte, la probabilità che fallisca diventa molto bassa
- questo fatto non ha solo un interesse teorico
- se sospettiamo che Eve ha trovato d, non basta cambiare l'esponente e – bisogna scegliere anche un nuovo N

modulo comune

- inoltre, si può pensare a una versione dell'RSA con una "trusted authority" che distribuisce le chiavi ai vari utenti
- in cui, per ogni utente, la chiave pubblica sia (N_u, e_u) e quella privata sia soltanto (N_u, d_u)
- dal momento che si può decifrare anche senza conoscere la fattorizzazione di N
- in questa situazione, non si può assegnare lo stesso N a diversi utenti
- per quanto visto ora, questo consentirebbe a ogni utente di violare i messaggi di ogni altro utente

RSA e fattorizzazione

- risolvere uno dei tre "problemi difficili" è computazionalmente equivalente a fattorizzare N
- questo non basta a dire che per violare bisogna fattorizzare N
- si potrebbe riuscire a decrittare senza calcolare l'esponente d

l'equivalenza si ha con il

RSA problem: dati e e N, calcolare le radici e-sime modulo N

non si sa se l'RSA problem è equivalente alla fattorizzazione

RSA problem e fattorizzazione

RSA problem: dati e e N, calcolare le radici e-sime modulo N

- se si sa fattorizzare N, si può risolvere l'RSA problem
- c'è un algoritmo polinomiale che calcola radici e-sime modulo p se $(e, p-1) = 1$
- si può mostrare (teorema cinese dei resti) che calcolare la radice e-sima di x mod p e mod $q \Rightarrow$ calcolare la radice e-sima di x mod $N = pq$
- per provare l'equivalenza basta provare una riduzione del tipo
  - algoritmo efficiente per il calcolo relle radici e-sime mod $N \Rightarrow$ algoritmo efficiente per fattorizzare N
- trovare questa riduzione è probabilmente il più importante problema aperto della PKC
- una tale riduzione esiste per $e = 2$ - radici quadrate. Questo si usa nel CS di Rabin

RSA – funzione trapdoor unidirezionale

- una trapdoor (botola) one-way function è una funzione unidirezionale che diventa facile da invertire, se si conosce un'informazione supplementare
- la nostra lo è: abbiamo visto che si può invertire la f.ne $f_e : x \to x^e \pmod N$ usando la funzione $f_d : y \to y^d \pmod N$, se $ed \equiv 1 \pmod{\phi(N)}$
- l' informazione supplementare è l'esponente d – difficile da ricavare dalle informazioni pubbliche

fattorizzazione

- la difficoltà di fattorizzare interi grandi non va comunque sopravvalutata
- nel 77, Rivest, Shamir e Adelman hanno proposto una sfida nella rubrica di Martin Gardner su *Scientific American*
- bisognava decifrare un testo cifrato con l'RSA-129 – con chiave pubblica $e = 9007$ e

```
N = 114381625757888867669235779976146
        612010218296721242362562561842935 7
        069352457338978305971235639587050 5
        898907514759929002687954354 1
```

- Rivest stimava che per fattorizzare N ci sarebbero voluti 40 quadrillioni di anni (1 quadrillione $=10^{15}$)
- il testo è stato decrittato nel 1994, da un team coordinato da Derek Atkins, Michael Graff, Arjen Lenstra, Paul Leyland usando il calcolo distribuito – il calcolo è durato sei mesi, ha coinvolto 1600 macchine
- il testo in chiaro era

  the magic words are squeamish ossifrage

- questo successo è dovuto essenzialmente al miglioramento degli algoritmi di fattorizzazione

algoritmi di fattorizzazione

n = lunghezza dell'intero N

- divisioni successive bisogna fare $< \sqrt{N}$ divisioni - è in $\mathcal{O}(2^{n/2})$; se $N = p \cdot q$ e p e q sono vicini, bisogna farle quasi tutte
- fattorizzazione alla Fermat idea - se ho x e $y \in \mathbb{N}$ tali che $N + y^2 = x^2$ – ho la fattorizzazione $N = (x + y)(x - y)$. L'algoritmo controlla per piccoli valori di y se $N + y^2$ è un quadrato perfetto.
- Esempio: sia $N = 5609$:
  $y = 1 \quad N + y^2 = 5610$ non è un quadrato
  $y = 2 \quad N + y^2 = 5613$ non è un quadrato
  $y = 3 \quad N + y^2 = 5618$ non è un quadrato
  $y = 4 \quad N + y^2 = 5625 = 75^2$
  quindi $5609 = 75^2 - 4^2 = (75 + 4)(75 - 4) = 79 \cdot 71$
  funziona meglio delle divisioni successive se N è il prodotto di due primi della stessa grandezza

algoritmi di fattorizzazione

n = lunghezza dell'intero N

- algoritmo di Lehman combina i due metodi precedenti – è in $\mathcal{O}(2^{n/3})$
- ρ di Pollard è in $\mathcal{O}(2^{n/4})$
- $p - 1$ di Pollard funziona bene se N ha un divisore primo p tale che $p - 1$ ha tutti i fattori primi piccoli
- crivello quadratico di Pomerance – è il più rapido - e il più usato - sotto le 100 cifre decimali
- general number field sieve (crivello generale) il più rapido in assoluto – molto complesso
- poi c'è l'algoritmo di Shor (1994) – in un computer quantistico, la fattorizzazione ha complessità polinomiale

scelta dei primi

per evitare che la fattorizzazione abbia successo, si scelgono primi forti

- p e q devono avere all'incirca la stessa lunghezza
- $p-1$ e $q-1$ devono avere un fattore primo grande altrimenti si può usare l'algoritmo $p-1$ di Pollard
- la lunghezza minima raccomandata per N è 1024 bit (ca 300 cifre decimali) – RSA-1024 – comincia a sembrare un po' poco
- con un N a 2048 bit (ca 600 cifre decimali) si sta tranquilli (?) fino al 2030
- sempre più spesso si arriva a 4096 bit

scelta di e

- si cerca di scegliere e non troppo grande e tale che nella scrittura binaria di e ci siano pochi 1
- e piccolo = cifratura più veloce
  - nell'alg. square and multiply bisogna calcolare pochi quadrati
- meno 1 = cifratura più veloce
  - il numero di 1 è collegato al numero di moltiplicazioni nell'alg. square and multiply
- valori usati sono $e = 3$ (inizialemente, ma dà problemi), $e = 2^{16} + 1 = 65537$
- per il valore $2^{16} + 1$ sono necessarie 17 moltiplicazioni - per un $b < \phi(N)$ casuale, la stima è di circa 1000 moltiplicazioni
- non vuol dire che anche d sarà piccolo

low exponent attack

- d non deve essere piccolo
- se $3d < N^{1/4}$ e $q < p < 2q$ c'è un attacco dovuto a Wiener
- permette di ricavare rapidamente d
- se N ha 1024 bit, d deve avere almeno 256 bit
- la decifratura è computazionalmente pesante - questo è un problema per esempio per le smartcard
- si può usare il teorema cinese del resto per velocizzare la decifratura

RSA – side channel attack

- sono attacchi all'implementazione dell'algoritmo, non all'algoritmo
- uno è il timing attack
- analogia: uno scassinatore che cerca di capire la combinazione di una cassaforte dal tempo impiegato a "girare le rotelle"
- per l'RSA, si cerca di capire qual è la scrittura binaria dell'esponente di decifratura d osservando il tempo impiegato a decifrare
- nell'algoritmo square and multiply, si fa una moltiplicazione solo quando nella scrittura binaria di d c'è un 1
- posso risalire al numero di 1 in d
- un altro è il power monitoring attack