RSA: sicurezza e attacchi

Sicurezza

legata alla difficoltà di fattorizzare un numero arbitrario molto grande

fattorizzare $\Rightarrow$ forzare RSA ✓

fattorizzare $\Leftarrow$ forzare RSA ?

calcolo della radice in modulo

$$m = \sqrt[e]{c} \mod n$$

difficile almeno quanto fattorizzare (n è composto)

Calcolare $\phi(n)$ direttamente da n è equivalente a fattorizzare n

Ricavare d direttamente da (n, e) sembra costoso quanto fattorizzare n

Come possiamo fattorizzare un intero n ?

La fattorizzazione è un problema difficile, ma non più come un tempo…

- Da un lato la potenza di calcolo aumenta, dall'altro gli algoritmi di fattorizzazione vengono raffinati
- Esistono algoritmi relativamente veloci di complessità subesponenziale
  - General Number Field Sieve (GNFS) richiede $O(2^{\sqrt{b \log b}})$, operazioni per fattorizzare un intero n, con b = [$\log_2$ n] + 1 bit
  - Un attacco bruteforce ne richiede O(n), i.e., $O(2^b)$
- Con la potenza di calcolo attuale, usando l'algoritmo GNFS è possibile fattorizzare semiprimi fino a circa 768 bit
- Per interi con una struttura particolare, esistono algoritmi di fattorizzazione particolarmente efficienti
- La fattorizzazione e il logaritmo discreto non sono problemi NP-hard, e si possono risolvere in tempo polinomiale su macchine quantistiche

RSA- scelta dei parametri

Scegliere p e q di almeno 1024 bit.

TDEA, AES (bit della chiave)	*RSA* e *DH* (bit del modulo)	*ECC* (bit dell'ordine)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

RSA- vincoli su p e q

- Scegliere p e q molto grandi (di almeno 1024 bit) per resistere agli attacchi bruteforce

- Sia $p - 1$ che $q - 1$ devono contenere un fattore primo grande (altrimenti n si fattorizza velocemente)

- gcd(p-1, q-1) deve essere piccolo

  conviene scegliere p e q tale che (p-1)/2 e (q-1)/2 siano co-primi

- **Non riusare uno dei primi per altri moduli!**

  $n_1 = p * q_1$

  $n_2 = p * q_2$

  allora $p = \gcd(n_1, n_2)$

RSA- vincoli su p e q

Scegliere p e q **non troppo vicini** tra loro

$n \sim p^2 \sim q^2$ e quindi anche $\sqrt{n}$ sarà vicino ai primi

basterà quindi un attacco bruteforce che cerca i fattori vicino alla $\sqrt{n}$

$$\left(\frac{p+q}{2}\right)^2 - n = \left(\frac{p-q}{2}\right)^2$$

$$1) \quad \frac{(p+q)}{2} > \sqrt{n}$$

$$2) \quad \frac{(p+q)^2}{4} - n \text{ è un quadrato perfetto}$$

Si scandiscono gli interi maggiori di $\sqrt{n}$ fino a trovare z t.c.

$$z^2 - n = w^2$$

si suppone

$$z = (p+q)/2 \qquad w = (p-q)/2$$

da cui

$$p = z + w \qquad q = z - w$$

la vicinanza tra p e q assicura che non dobbiamo allontanarci troppo da $\sqrt{n}$ per trovare p e q

Attacchi con esponenti bassi

- esponenti e e d bassi sono attraenti perché accellerano cifratura o decifrazione
- ovviamente d dovrebbe essere scelto sufficientemente grande, per evitare attacchi bruteforce
- ATTENZIONE

  se m ed e sono così piccoli che $m^e < n$, allora risulta **facile** trovare la radice e-esima di c, poiché $c = m^e$

  non interviene la riduzione in modulo!

Attacchi a tempo (DH, RSA)

- Si basano sul tempo di esecuzione dell'algoritmo di decifrazione
- **IDEA:** determinare d analizzando il tempo impiegato per decifrare
  - Quando viene eseguita l'esponenziazione modulare, si esegue una moltiplicazione ad ogni iterazione, più un'ulteriore moltiplicazione modulare per ciascun bit uguale a 1 in d
  - **Rimedio**: aggiungere ritardo causale per confondere l'attaccante