Crittosistema a chiave pubblica

file pubblico

utente	chiave pubblica
A	kpub
…	…

Cifratura

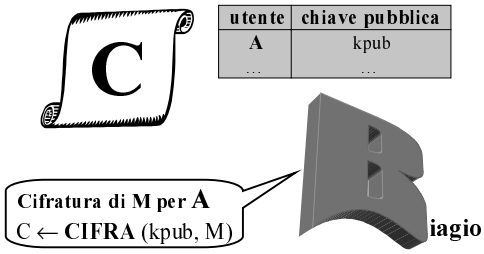
file pubblico

utente	chiave pubblica
A	kpub
…	…

Cifratura

file pubblico

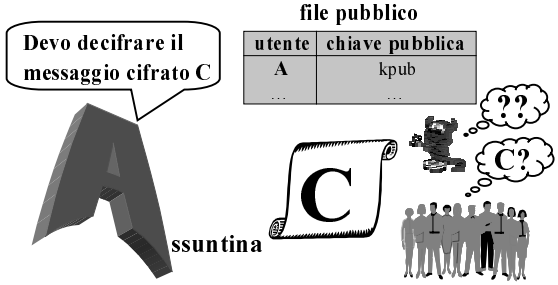
utente	chiave pubblica
A	kpub
…	…



Decifratura

file pubblico

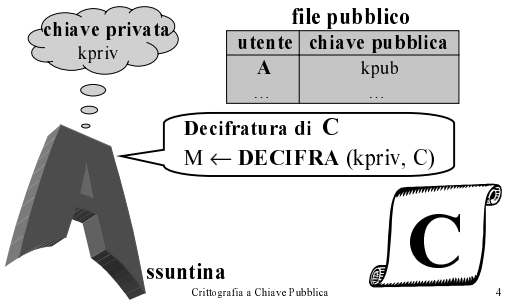
utente	chiave pubblica
A	kpub
…	…



Decifratura

file pubblico

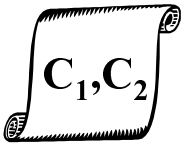
utente	chiave pubblica
A	kpub
…	…



Crittografia a chiave pubblica ed a chiave privata

- Vantaggi della crittografia a chiave pubblica
  - Chiavi private mai trasmesse
  - Possibile la firma digitale
- Vantaggi della crittografia a chiave privata
  - Molto più veloce (ad es., DES è ≥100 volte più veloce di RSA, in hardware tra 1.000 e 10.000 volte)
  - Sufficiente in diverse situazioni (ad esempio, applicazioni per singolo utente)

Digital Envelope



file pubblico

utente	chiave pubblica
A	kpub
…	…

Cifratura di M per A
k ← genera chiave sessione
C_1 ← **CIFRA** (kpub, k)
C_2 ← **E** (k, M)

iagio

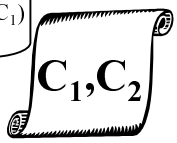
Digital Envelope: Decifratura

chiave privata
kpriv

file pubblico

utente	chiave pubblica
A	kpub
…	…

Decifratura di C_1,C_2
k ← **DECIFRA** (kpriv, C_1)
M ← **D** (k, C_2)



Digital Envelope: vantaggi

- Chiave di sessione solo per uno o pochi messaggi
- Molto più veloce della sola crittografia a chiave pubblica

Teoria dei Numeri

Crittosistemi a chiave pubblica più comuni basati sulla Teoria dei Numeri

"… both Gauss and lesser mathematicians may be justified in rejoicing that there is one science at any rate, and that their own, whose very remoteness from ordinary human activities keep it gentle and clean."
G. H. Hardy, *A Mathematician's Apology,* 1940

RSA

- Proposto da **R**ivest, **S**hamir, ed **A**dleman, nel 1978
- Sicurezza basata sulla difficoltà di fattorizzare

Chiavi RSA

chiave privata
(n,d)

file pubblico

utente	chiave pubblica
A	(n,e)
…	…

n = pq
p,q primi

ed = 1 mod (p-1)(q-1)

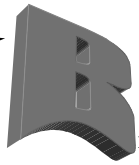
ssuntina

Cifratura RSA

file pubblico

utente	chiave pubblica
A	(n,e)
…	…

Devo cifrare il messaggio M ed inviarlo ad A



iagio

Cifratura RSA

C

utente	chiave pubblica
A	(n,e)
…	…

Cifratura di M per A

$C \leftarrow M^e \bmod n$

Decifratura RSA

Devo decifrare il messaggio cifrato C

utente	chiave pubblica
A	(n,e)
…	…

C

Decifratura RSA

chiave privata (n,d)

file pubblico

utente	chiave pubblica
A	(n,e)
…	…

Decifratura di C

$M \leftarrow C^d \bmod n$

C

ssuntina

"Piccolo" esempio: Chiavi RSA

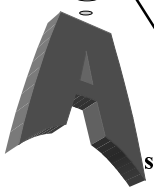
chiave privata (n=3337, d=1019)

file pubblico

utente	chiave pubblica
A	(n = 3337, e = 79)
…	…

3337 = 47 ·71
p = 47, q = 71

ed = 79 ·1019 = 1 mod 3220
(p-1)(q-1) = 46 ·70 = 3220



ssuntina

Esempio: Cifratura RSA

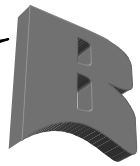
1570

file pubblico

utente	chiave pubblica
A	(n = 3337, e = 79)
…	…

Cifratura di M = 688 per A

$1570 \leftarrow 688^{79} \bmod 3337$



iagio

Esempio: Decifratura RSA

chiave privata (n=3337, d=1019)

file pubblico

utente	chiave pubblica
A	(n = 3337, e = 79)
…	…

Decifratura di C = 1570
$688 \leftarrow 1570^{1019} \bmod 3337$

1570

ssuntina

Correttezza decifratura RSA

$C^d \bmod n = (M^e)^d \bmod n$
$= M^{de} \bmod n$ — ed = 1 mod (p-1)(q-1)
$= M^{1+r(p-1)(q-1)} \bmod n$
$= M \bmod n$
$= M$ — Teorema di Eulero: $x \in Z_n^* \Rightarrow x^{(p-1)(q-1)} = 1 \bmod n$

poichè $0 \le M < n$

Prova per tutti gli x mediante il teorema del resto cinese

Funzione di Eulero

- $\phi(n)$ = cardinalità di $\mathbf{Z_n^* = \{ x \mid 0<x<n \text{ tali che } gcd(x,n)=1 \}}$
- $\phi(p) = p-1$ se p primo
- $\phi(pq) = (p-1)(q-1)$ se p,q primi
- $\phi(n) = n \cdot \left(1-\frac{1}{p_1}\right) \cdot \left(1-\frac{1}{p_2}\right) \cdots \left(1-\frac{1}{p_k}\right)$

  fattorizzazione $n = p_1^{e_1} p_2^{e_2} \ldots p_k^{e_k}$, p_i primo, $e_i \ge 0$
- **Teorema di Eulero**: $x \in Z_n^* \Rightarrow x^{\phi(n)} = 1 \bmod n$

Efficienza delle computazioni

Come effettuare le computazioni?

- Elevazione a potenza modulare
- Generazione numeri primi
- Generazione e,d
  - generazione di e
  - $d \leftarrow e^{-1} \bmod (p-1)(q-1)$

Elevazione a potenza modulare Metodo naive

Calcolo di $\mathbf{x^y \bmod z}$

```
Potenza_Modulare_naive (x, y, z)
a ← 1
for i = 1 to y do
        a ← (a · x) mod z
return a
```


Elevazione a potenza modulare Metodo naive

Calcolo di $\mathbf{x^y \bmod z}$

```
Potenza_Modulare_naive (x, y, z)
a ← 1
for i = 1 to y do
        a ← (a · x) mod z
return a
```

Se y è di 512 bit, occorrono $\approx 2^{512}$ operazioni

Elevazione a potenza modulare
Metodo left-to-right

Calcolo di $x^y \bmod z$ $\quad y = y_0 2^0 + y_1 2^1 + \dots + y_t 2^t$

Idea: $y = y_0 + 2(y_1 + \dots + 2(y_{t-1} + 2y_t)))))$

$x^y = x^{y_0} (\dots (x^{y_{t-1}} (x^{y_t})^2)^2 \dots)^2$

Esempio: x^{40}

$40 = 0\cdot 2^0 + 0\cdot 2^1 + 0\cdot 2^2 + 1\cdot 2^3 + 0\cdot 2^4 + 1\cdot 2^5$

$40 = 0 + (2\,(0 + 2\,(0 + (2\,(1 + 2\,(0 + 2\cdot 1))))))$

Elevazione a potenza modulare
Metodo left-to-right

Calcolo di $x^y \bmod z$ $\quad y = y_0 2^0 + y_1 2^1 + \dots + y_t 2^t$

Idea: $y = y_0 + 2(y_1 + \dots + 2(y_{t-1} + 2y_t)))))$

$x^y = x^{y_0} (\dots (x^{y_{t-1}} (x^{y_t})^2)^2 \dots)^2$

Esempio: x^{40}

$40 = 0\cdot 2^0 + 0\cdot 2^1 + 0\cdot 2^2 + 1\cdot 2^3 + 0\cdot 2^4 + 1\cdot 2^5$

$40 = 0 + (2\,(0 + 2\,(0 + (2\,(1 + 2\,(0 + 2\cdot 1))))))$

$x^{40} = x^0\ (\ x^0\ \ (x^0\ \ (x^1\ \ (x^0\ (x^1)^2)^2)^2)^2)^2$

Elevazione a potenza modulare
Metodo left-to-right

Calcolo di $x^y \bmod z$ $\quad y = y_0 2^0 + y_1 2^1 + \dots + y_t 2^t$

Idea: $y = y_0 + 2(y_1 + \dots + 2(y_{t-1} + 2y_t)))))$

$x^y = x^{y_0} (\dots (x^{y_{t-1}} (x^{y_t})^2)^2 \dots)^2$

Potenza_Modulare (x, y, z) **left-to-right**

$a \leftarrow 1$
for i = t **downto** 0 **do**
  $a \leftarrow (a \cdot a) \bmod z$
  if $y_i = 1$ **then** $a \leftarrow (a \cdot x) \bmod z$
return a

Elevazione a potenza modulare
Metodo right-to-left

Calcolo di $x^y \bmod z$ $\quad y = y_0 2^0 + y_1 2^1 + \dots + y_t 2^t$

Idea: $x^y = (x^{2^0})^{y_0} \cdot (x^{2^1})^{y_1} \cdot \dots \cdot (x^{2^t})^{y_t}$

Esempio: x^{40}

$40 = 0\cdot 2^0 + 0\cdot 2^1 + 0\cdot 2^2 + 1\cdot 2^3 + 0\cdot 2^4 + 1\cdot 2^5$

$x^{40} = (x^1)^0 \cdot (x^2)^0 \cdot (x^4)^0 \cdot (x^8)^1 \cdot (x^{16})^0 \cdot (x^{32})^1$

Elevazione a potenza modulare
Metodo right-to-left

Calcolo di $x^y \bmod z$ $\quad y = y_0 2^0 + y_1 2^1 + \dots + y_t 2^t$

Idea: $x^y = (x^{2^0})^{y_0} \cdot (x^{2^1})^{y_1} \cdot \dots \cdot (x^{2^t})^{y_t}$

Esempio: x^{40}

$40 = 0\cdot 2^0 + 0\cdot 2^1 + 0\cdot 2^2 + 1\cdot 2^3 + 0\cdot 2^4 + 1\cdot 2^5$

$x^{40} = (x^1)^0 \cdot (x^2)^0 \cdot (x^4)^0 \cdot (x^8)^1 \cdot (x^{16})^0 \cdot (x^{32})^1$

$x^1 \quad x^2 \quad x^4 \quad x^8 \quad x^{16} \quad x^{32}$

Elevazione a potenza modulare
Metodo right-to-left

Calcolo di $x^y \bmod z$ $\quad y = y_0 2^0 + y_1 2^1 + \dots + y_t 2^t$

Idea: $x^y = (x^{2^0})^{y_0} \cdot (x^{2^1})^{y_1} \cdot \dots \cdot (x^{2^t})^{y_t}$

Esempio: x^{40}

$40 = 0\cdot 2^0 + 0\cdot 2^1 + 0\cdot 2^2 + 1\cdot 2^3 + 0\cdot 2^4 + 1\cdot 2^5$

$x^{40} = (x^1)^0 \cdot (x^2)^0 \cdot (x^4)^0 \cdot (x^8)^1 \cdot (x^{16})^0 \cdot (x^{32})^1$

$x^{40} =$ (=1) (=1) (=1) x^8 (=1) $\cdot\ x^{32}$

Elevazione a potenza modulare

Metodo right-to-left

Calcolo di $x^y \bmod z$ $\quad y = y_0 2^0 + y_1 2^1 + \ldots + y_t 2^t$

Idea: $x^y = (x^{2^0})^{y_0} \cdot (x^{2^1})^{y_1} \cdot \ldots \cdot (x^{2^t})^{y_t}$

```
Potenza_Modulare (x, y, z)
if y = 0 then return 1
X ← x; P ← 1
if y_0 = 1 then X ← x
for i = 1 to t do
    X ← X ·X mod z
    if y_i=1 then P ← P ·X mod z
return P
```

rigth-to-left square-and-multiply

Elevazione a potenza modulare

Metodo right-to-left

```
Potenza_Modulare (x, y, z)
if y = 0 then return 1
X ← x; P ← 1
if y_0 = 1 then X ← x
for i = 1 to t do
    X ← X ·X mod z
    if y_i=1 then P ← P ·X mod z
return P
```

$5^{596} \bmod 1234$

i	0	1	2	3	4	5	6	7	8	9
y_i	0	0	1	0	1	0	1	0	0	1
X	5	25	625	681	1011	369	421	779	947	925
P	1	1	625	625	67	67	1059	1059	1059	1013

Decifratura RSA

- Pentium 90MHz, toolkit BSAFE 3.0

modulo	512 bit	1024 bit
throughput kbit/s	21,6	7,4

- RSA hardware [1993]

modulo	512 bit	970 bit
throughput kbit/s	300	185

Algoritmo di Euclide

Descritto negli *Elementi* di Euclide (circa 300 A. C.)

```
Euclide (a,b)
if b = 0 then return a
         else return Euclide (b, a mod b)
```

Teorema della ricorsione del gcd
Per tutti gli interi $a \geq 0$ e $b > 0$
$\gcd(a,b) = \gcd(b, a \bmod b)$

Algoritmo di Euclide: Esempi

Euclide (30,21) = Euclide (21,9)
= Euclide (9,3)
= Euclide (3,0) = 3

Euclide (4864,3458) = Euclide (3458,1406)
= Euclide (1406,646)
= Euclide (646,114)
= Euclide (114,76)
= Euclide (76,38)
= Euclide (38,0) = 38

Algoritmo di Euclide: complessità

- Assumiamo $a \geq b$
- Al massimo $\log b$ chiamate
- Per ogni chiamata $O((\log a)^2)$ operazioni su bit
- Totale: al massimo $O((\log a)^3)$ operazioni su bit
- E' possibile mostrare che Euclide (a,b) richiede al massimo $O((\log a)^2)$ operazioni su bit

Algoritmo di Euclide Esteso

```
Euclide-esteso (a,b)
if b = 0 then return (a, 1, 0)
(d′, x′, y′) ← Euclide-esteso (b, a mod b)
(d, x, y) ← (d′, y′, x′- ⌊a/b⌋ y′)
return (d, x, y)
```

- Computa interi d, x, y tali che d = gcd(a,b) = ax+by
- Stesso running time asintotico di Euclide (a,b)

Algoritmo di Euclide (iterativo)

```
Euclide_iterativo (a,b)
while b ≠ 0 do
        r ← a mod b; a ← b; b ← r;
return a
```

Esercizio: versione iterativa di Euclide-esteso (a,b)

Soluzione di ax≡b (mod n)

- Ha soluzioni se e solo se g|b g = gcd(a,n)
- Se g|b ci sono esattamente g distinte soluzioni mod n:

$$x'\frac{b}{g}+i\frac{n}{g} \qquad \text{per } i = 0,1,\ldots, g-1$$

dove g = ax′+ny (da Euclide-esteso (a,n))

Soluzione di ax≡1 (mod n)

- Ha soluzioni se e solo se gcd(a,n) =1
- Se gcd(a,n) =1 l'unica soluzione mod n è:

x′

dove 1 = ax′+ny (da Euclide-esteso (a,n))

- Tale soluzione viene denotata con a^{-1} mod n

Generazione chiavi

1. Input L (lunghezza modulo)
2. Genera 2 primi di lunghezza L/2
3. n ← p ·q
4. Scegli a caso e
5. **If** gcd (e, (p-1)(q-1)) = 1
   then d ← e^{-1} mod (p-1)(q-1)
   else goto 4.

Generazione chiavi (comunemente usata in pratica)

1. Input L (lunghezza modulo)
2. e ← 3 oppure e ← 2^{16}+1 (= 65.537) **solo due 1 in binario!**
3. Genera 2 primi di lunghezza L/2
4. n ← p ·q
5. **If** gcd (e, (p-1)(q-1)) = 1
   then d ← e^{-1} mod (p-1)(q-1)
   else goto 3.

Generazione di un primo 'grande'

1. Genera a caso un dispari p di grandezza appropriata
2. Testa se p è primo
3. Se p è composto, go to **1.**

Variante con sequenza di ricerca

1. Genera a caso un dispari p di grandezza appropriata
2. Testa se p è primo
3. Se p è composto, $p \leftarrow p+2$, go to **2.**

Distribuzione dei numeri primi

- $\pi(x)$ = numero di primi in [2,x]
- Teorema dei numeri primi: $\lim_{x\to\infty} \frac{\pi(x)}{x/\ln x} = 1$
- Esempio: $\pi(10^{10}) = 455.052.511$

  $10^{10}/\ln 10^{10} \approx 434.294.481{,}9$ (4% in meno)
- Per $x \geq 17$ $\quad \frac{x}{\ln x} < \pi(x) < 1.25506\frac{x}{\ln x}$

Scelta di un primo di 512 bit

- Scelto un intero in $[2,2^{512}]$ la probabilità che sia primo è circa 1 su $\ln 2^{512}$ $(\ln 2^{512} \approx 354.89)$
- Numero medio di tentativi ≈ 354.89
- Se si scelgono solo numeri dispari dimezza ≈ 177.44
- Se si scelgono dispari in $[2^{511},2^{512}]$ la probabilità è

$$\approx \left(\frac{2^{512}}{\ln 2^{512}} - \frac{2^{511}}{\ln 2^{511}}\right)\frac{1}{2^{511}/2} \approx \frac{1}{177.79}$$

Scelta di un primo di 512 bit

- Scelto un intero in $[2,2^{512}]$ la probabilità che sia primo è circa 1 su $\ln 2^{512} \approx 354.89$
- Numero medio di tentativi ≈ 354.89
- Se si scelgono solo numeri dispari dimezza ≈ 177.44
- Se si scelgono dispari in $[2^{511},2^{512}]$ la probabilità è

$$\approx \left(\frac{2^{512}}{\ln 2^{512}} - \frac{2^{511}}{\ln 2^{511}}\right)\frac{1}{2^{511}/2} \approx \frac{1}{177.79}$$

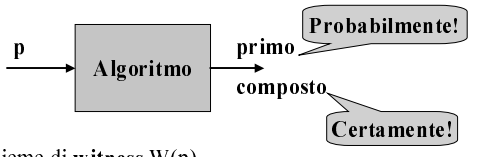
- Per scegliere un numero di 512 bit ($\approx$154 digit):

$$\underbrace{1\ldots\ldots\ldots 1}_{\text{510 bit scelti a caso}}$$

Test di primalità

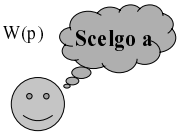
- Il miglior algoritmo deterministico è una variante di Cohen e H. Lenstra del test di Adleman, Pomerance e Rumely [1983]
- Complessità $(\lg p)^{O(\lg\lg\lg p)}$

Test di primalità probabilistici



Insieme di **witness** W(p)

- dato $a \in [0, p-1]$ è facile verificare se $a \in W(p)$
- se p è primo allora W(p) è vuoto
- se p è composto allora $|W(p)| \geq p/2$



Diminuzione della probabilità di errore

- Probabilità di errore (n è composto ma viene dichiarato primo) di tale test $\leq 1/2$
- Se il test viene ripetuto indipendentemente t volte allora la probabilità di errore $\leq (1/2)^t$

Test di primalità probabilistici

- Test di Solovay-Strassen
  - Pubblicato nel 1977
  - Probabilità di errore $\leq (1/2)^t$
- Test di Miller-Rabin
  - Il più usato in pratica
  - Il più veloce
  - Probabilità di errore $\leq (1/4)^t$

Test di Solovay-Strassen

- $(a|n)$ simbolo di Jacobi
- **Criterio di Eulero**: Sia n un primo dispari.
  $\gcd(a,n) = 1 \Rightarrow a^{(n-1)/2} = (a|n) \mod n$
- Sia n un numero composto dispari.
  $\left|\{a : \gcd(a,n) = 1 \text{ and } a^{(n-1)/2} = (a|n) \mod n\}\right| \leq \phi(n)/2$
- Insieme di *witness di Eulero*
  $W_E(n) = \{a : \gcd(a,n)>1 \text{ oppure } a^{(n-1)/2} \neq (a|n) \mod n\}$

Simbolo di Jacobi

- $(a|n) = (a|p_1)^{e_1} (a|p_2)^{e_2} \ldots (a|p_k)^{e_k}$

$$(a|p) = \begin{cases} 0 & \text{se p divide a} \\ 1 & \text{se a è un quadrato mod p} \\ -1 & \text{se a non è un quadrato mod p} \end{cases}$$

Simbolo di Legendre
p primo
$(a|p) = a^{(p-1)/2} \mod p$

fattorizzazione dispari $n = p_1^{e_1} p_2^{e_2} \ldots p_k^{e_k}$, p_i primo, $e_i \geq 0$

- Può essere computato con $O((\log n)^2)$ operazioni su bit
  - $(a|n) = 0 \Leftrightarrow \gcd(a,n) > 1$
  - $a = b \mod n \Rightarrow (a|n) = (b|n)$
  - $(ab|n) = (a|n)(b|n)$
  - $(2|n) = (-1)^{(n^2-1)/8}$
  - $(m|n) = (n|m)(-1)^{(n-1)(m-1)/4}$

Simbolo di Jacobi

Può essere computato con $O((\log n)^2)$ operazioni su bit
- $a = b \mod n \Rightarrow (a|n) = (b|n)$
- $(ab|n) = (a|n)(b|n)$
- $(2|n) = (-1)^{(n^2-1)/8}$
- $(m|n) = (n|m)(-1)^{(n-1)(m-1)/4}$

Esempio: $(158|235) = (2|235)(79|235)$
$= (-1)(235|79)(-1)^{(78)(234)/4}$
$= (77|79)$
$= (79|77)(-1)^{(76)(78)/4}$
$= (2|77)$
$= -1$

Test di Miller-Rabin

- Insieme di *witness di Miller-Rabin*

$n-1 = 2^s r$ con r dispari

$W_{MR}(n) = \{a : a^r \neq 1 \mod n \text{ and } a^{2^j r} \neq -1 \mod n \text{ per ogni } j \in [0,s-1]\}$

- Se n è un primo dispari
  $W_{MR}(n)$ è vuoto
- Se n è un numero composto dispari $(n \neq 9)$
  $|W_{MR}(n)| \geq (3/4)\cdot\phi(n)$

Witness di Miller-Rabin n=91

$91 = 7 \cdot 13$

$91-1 = 2^1 \cdot 45$

- W*itness di Miller-Rabin*

  $W_{MR}(91) = \{a : a^{45} \neq 1 \bmod 91$ and $a^{2^j \cdot 45} \neq -1 \bmod 91$ per ogni $j \in [0,0]\}$

- Se n è un numero composto dispari

  $|W_{MR}(n)| \geq (3/4) \cdot \phi(n)$

- **Strong liars**: elementi in $\{1,2,\ldots,90\} / W_{MR}(91)$

  $\{1,9,10,12,16,17,22,29,38,53,62,69,74,75,79,81,82,90\}$

  Sono $18 = \phi(91)/4$ elementi

Witness di Miller-Rabin n=105

$105 = 3 \cdot 5 \cdot 7$

$105-1 = 2^3 \cdot 13$

- W*itness di Miller-Rabin*

  $W_{MR}(105) = \{a : a^{13} \neq 1 \bmod 105$ and $a^{2^j \cdot 13} \neq -1 \bmod 105$ per ogni $j \in [0,2]\}$

- **Strong liars**: elementi in $\{1,2,\ldots,104\} / W_{MR}(105)$

  $\{1,104\}$

  Sono solo $2 < 12 = \phi(105)/4$ elementi

In genere strong liars molto meno di $\phi(n)/4$

Fattorizzazione

Dato n calcolare l'unica fattorizzazione

$$n = p_1^{e_1} p_2^{e_2} \ldots p_k^{e_k}$$

con p_i primo ed $e_i \geq 0$

Splitting di n: calcolare due interi a,b >1 (fattori non-triviali) tali che $n = a \cdot b$

Fattorizzazione: un semplice algoritmo

Calcolo di un fattore primo:

Per tutti i primi p in $[2,\sqrt{n}]$

Se $p|n$ allora p è fattore di n

Complessità caso peggiore $\Theta(\sqrt{n})$ (accade se n = pq)

Se n ha 512 bit allora $\sqrt{n} \approx 2^{256}$

Fattorizzazione: complessità algoritmi

Complessità di tempo sub-esponenziale in media

$$L_q[a,c] = O(e^{(c+o(1))(\ln q)^a (\ln\ln q)^{1-a}})$$

con $c > 0$ ed $0 < a < 1$

- Algoritmo basato su curve ellittiche: $L_n[1/2, 1]$
- **Quadratic sieve**: $L_n[1/2, 1]$
- **General Number field sieve**: $L_n[1/3, 1]$ **il più veloce**

Quadratic sieve in pratica

- **Quadratic sieve** è migliore dell'algoritmo basato su curve ellittiche
- Implementazioni del **Quadratic sieve**

1 mips per anno $\approx 3 \cdot 10^{13}$ istruzioni

anno	numero digit	mips per anno
1984	71	0.01
1988	106	140
1993	120	825
1994	129	5000

RSA-129 1600 computer per 8 mesi factoring by e-mail

= 425 bit

General Number field sieve in pratica

- **General Number field sieve** è migliore del **Quadratic sieve** solo per interi di almeno 110-120 cifre decimali
- **RSA-130**, fattorizzato nel 1996 con 500 mips per anno
- **RSA-140**, fattorizzato il 2 febbraio 1999, elapsed time 9 settimane
- **RSA-155**, fattorizzato il 22 agosto 1999, con 8000 mips per anno, elapsed time 7.4 mesi

160	175-400 MHz SGI and Sun workstations
8	250 MHz SGI Origin 2000 processors
120	300-450 MHz Pentium II PCs
4	500 MHz Digital/Compaq boxes

512 bit!

Calcolo di fattori "piccoli"

Algoritmi per calcolare un fattore p di n:

- Algoritmo **Rho di Pollard**: tempo medio $O(\sqrt{p})$
- Algoritmo basato su curve ellittiche: tempo medio

$$L_p[1/2, \sqrt{2}] = O(e^{(\sqrt{2}+o(1))(\ln q)^{1/2}(\ln\ln q)^{1/2}})$$

I numeri più difficili da fattorizzare sono del tipo $n = pq$ con p,q primi della stessa lunghezza

Una strategia generale per fattorizzare

Cerca fattori "piccoli"

- Prova divisione per alcuni piccoli primi (2,3,5,7,…)
- Prova algoritmo **Rho di Pollard**
- Prova algoritmo basato su curve ellittiche

Cerca fattori "grandi"

- Prova **Quadratic sieve** oppure **General Number field sieve**

Che modulo scegliere?

- Ad oggi, i numeri più difficili da fattorizzare sono del tipo $n = p \cdot q$ con p,q primi della stessa lunghezza
- … e di almeno (per essere tranquilli!)
  - 768 bit (= 230 digit) per uso personale
  - 1024 bit per le aziende
  - 2048 per chiavi "importanti" ad esempio **Autorità di Certificazione**

Intrattabilità della fattorizzazione

La sicurezza di molte tecniche crittografiche si basa sulla intrattabilità della fattorizzazione:

- crittosistema RSA, Rabin
- firme digitali RSA
- …

Sicurezza di RSA

Se  potesse fattorizzare…

1. Fattorizza n
2. Computa $(p-1)(q-1)$
3. Computa $d \leftarrow e^{-1} \bmod (p-1)(q-1)$

Sicurezza di RSA

Se  potesse computare $\varphi(n)=(p-1)(q-1)$

potrebbe calcolare $d \leftarrow e^{-1} \bmod (p-1)(q-1)$

Sicurezza di RSA

Se  potesse computare $\varphi(n)=(p-1)(q-1)$,

potrebbe calcolare $d \leftarrow e^{-1} \bmod (p-1)(q-1)$

$$\begin{cases} n = pq \\ \varphi(n) = (p-1)(q-1) \end{cases}$$ sostituendo $p = n/q$

$p^2 - (n-\varphi(n)+1)p + n = 0$

Due soluzioni: p,q

Sicurezza di RSA

Se  potesse computare $\varphi(n) = (p-1)(q-1)$,

potrebbe calcolare $d \leftarrow e^{-1} \bmod (p-1)(q-1)$

$$\begin{cases} n = pq \\ \varphi(n) = (p-1)(q-1) \end{cases}$$ sostituendo $p = n/q$

$p^2 - (n-\varphi(n)+1)p + n = 0$

$$\begin{cases} 84.773.093 = pq \\ 84.754.668 = (p-1)(q-1) \end{cases}$$ $p^2 - 18.426\,p + 84.773.093 = 0$

radici: 9539 e 8887

Sicurezza di RSA

Se  potesse computare d …

Sicurezza di RSA

Se  potesse computare d … ma questo è computazionalmente equivalente a fattorizzare!

Sicurezza di RSA

Se  potesse computare d … ma questo è computazionalmente equivalente a fattorizzare!

Un algoritmo che computa d (con input n,e) può essere usato come oracolo in un algoritmo ***Las Vegas*** che fattorizza n con probabilità $\geq 1/2$

Piccoli messaggi

- Posso cifrare un solo digit 0,1,… ,9 con RSA ?

$C \leftarrow M^e \bmod n$

Esempio: $125 \leftarrow 5^3 \bmod 6.012.707$

Piccoli messaggi

- Posso cifrare un solo digit 0,1,… ,9 con RSA ?

$C \leftarrow M^e \bmod n$

Esempio: $125 \leftarrow 5^3 \bmod 6.012.707$

- Se $M < n^{1/e}$ allora
  - uso di Digital Envelope
  - *salting* del messaggio M

Proprietà moltiplicativa di RSA

Proprietà di omomorfismo

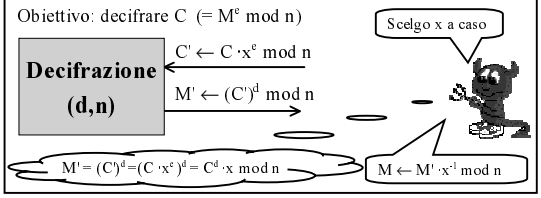
$C_1 = M_1^e \bmod n$
$C_2 = M_2^e \bmod n$
$\Big\} \quad (M_1M_2)^e = M_1^e M_2^e = C_1C_2 \bmod n$

Proprietà moltiplicativa di RSA

$C_1 = M_1^e \bmod n$
$C_2 = M_2^e \bmod n$
$\Big\} \quad (M_1M_2)^e = M_1^e M_2^e = C_1C_2 \bmod n$

Chosen-ciphertext attack adattivo

Obiettivo: decifrare C $(= M^e \bmod n)$



Attacchi ad RSA

- Piccolo esponente pubblico (ad es., e = 3)
  - Stesso messaggio inviato a più utenti
- Attacchi ad implementazioni:
  - Timing Attack
  - Random Faults
  - Attacco di Bleichenbacher [1998] su PKCS 1 (vecchia versione): uso del Web browser come oracolo!

Informazioni parziali per RSA

Dato C **$C = M^e \bmod n$**

- potrebbero esserci informazioni parziali sul messaggio M "facili" da ottenere (senza dover decifrare C !)
- e … "difficili" ?

(*hard bit*: ultimi $c \cdot \log\log n$ bit)

Un bit "facile" per RSA

$C = M^e \bmod n$

E' facile calcolare il simbolo di Jacobi del testo in chiaro

$$(C|n) = (M^e|n) = (M|n)^e = (M|n)$$

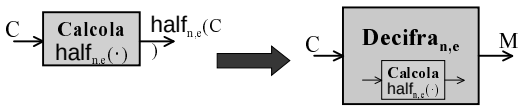
Un bit "difficile" per RSA

$$\text{half}_{n,e}(C) = \begin{cases} 0 & \text{se } M < n/2 \\ 1 & \text{se } M > n/2 \end{cases}$$

Un bit "difficile" per RSA

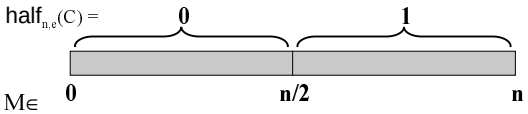
$$\text{half}_{n,e}(C) = \begin{cases} 0 & \text{se } M < n/2 \\ 1 & \text{se } M > n/2 \end{cases}$$

Calcolare $\text{half}_{n,e}(\cdot)$ è computazionalmente equivalente ad invertire RSA



Un bit "difficile" per RSA

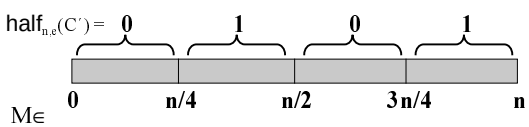
$$\text{half}_{n,e}(C) = \begin{cases} 0 & \text{se } M < n/2 \\ 1 & \text{se } M > n/2 \end{cases}$$



Un bit "difficile" per RSA

$$\text{half}_{n,e}(C) = \begin{cases} 0 & \text{se } M < n/2 \\ 1 & \text{se } M > n/2 \end{cases}$$

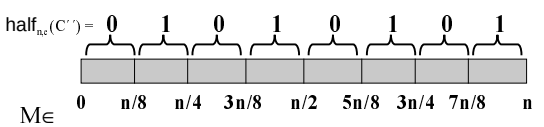
$C' \leftarrow C \cdot (2^e \bmod n)$ $\quad C' = (2 \cdot M)^e \bmod n$



Un bit "difficile" per RSA

$$\text{half}_{n,e}(C) = \begin{cases} 0 & \text{se } M < n/2 \\ 1 & \text{se } M > n/2 \end{cases}$$

$C'' \leftarrow C \cdot (4^e \bmod n)$ $\quad C'' = (4 \cdot M)^e \bmod n$



Un bit "difficile" per RSA

- Ricerca binaria
- Dopo log n chiamate ad $\text{half}_{n,e}(\cdot)$ l'intervallo consta di un solo valore

Predicato parità

$C = M^e \bmod n$

$\text{parità}_{n,e}(C)$ = bit meno significativo di M

I predicati $\text{parità}_{n,e}(\cdot)$ e $\text{half}_{n,e}(\cdot)$ sono computazionalmente equivalenti

$\text{parità}_{n,e}(C) = \text{half}_{n,e}(y)$

$y = C \cdot (2^{-1})^e \bmod n$
$= (M \cdot 2^{-1})^e \bmod n$

Crittografia probabilistica

Testo in chiaro $M = M_1 M_2 M_3 \ldots$
Testo cifrato $C = C_1 \; C_2 \; C_3 \ldots$

$M_i = \text{predicato\_difficile}\,(C_i)$

Alcuni Crittosistemi a chiave pubblica

- RSA [1977] (fattorizzazione)
- Merkle-Hellman [1978] (zaino 0-1) rotto!
  - Molte varianti… rotte! Resiste Chor-Rivest [1988]
- Rabin [1979] (fattorizzazione)
- McEliece [1978] (decodifica codici lineari)
- El-Gamal [1984] (logaritmo discreto)
- Uso di curve ellittiche [1985]
  curve iperellittiche [1989]
  automi cellulari [1985]

Funzioni one-way

"Facili" da calcolare e

"difficili" da invertire

Funzioni one-way trapdoor

"Facili" da calcolare e

"difficili" da invertire

… a meno che si conosca una "trapdoor"

Alcuni brevetti software (USA)

numero	oggetto	inventore	scadenza
3.962.539	DES	Ehrsam ed al.	1993
4.200.770	Diffie-Hellman	Hellman, Diffie, Merkle	1997
4.218.582	Critt. chiave pubblica	Helman, Merkle	1997
4.424.414	Pohling-Hellman	Hellman, Pohling	3 gen 2001
4.405.829	RSA	Rivest, Shamir, Adleman	20 sett 2000
4.748.668	Shamir-Fiat	Shamir, Fiat	2005
4.850.017	Vettori di controllo	Matyas, Meyer, Bracht	2006
5.140.634	Guillou-Quisquater	Guillou-Quisquater	2009
5.231.668	DSA	Kravitz	2010
5.214.703	IDEA	Massey-Lai	2010
5.276.737	Fair Cryptosystems	Micali	2011

Crittografia a Chiave Pubblica 90

Export dagli USA

International Traffic in Arms Regulations (ITAR) [1989]

- United States Munitions List
- Strong encryption
  - RSA ≥ 512 bit
  - cifrari a blocco (DES, IDEA, RC6,…) ≥40 bit
- eccetto se limitato a
  - controllo accessi (ad es., ATM)
  - autenticazione dati (ad es., MAC, firme)

non deve essere facilmente utilizzabile per cifrare!

- permesso dal *Department of Commerce*,

ad es. Cybercash, cifratura RSA 768 bit per transazioni finanziarie

Crittografia a Chiave Pubblica 91