

# Crittografia a chiave pubblica

Corso di Sicurezza dei Dati  
a.a. 2020-21

**Alfredo De Santis**

Dipartimento di Informatica  
Università di Salerno

**ads@unisa.it**

**<http://www.di-srv.unisa.it/~ads>**

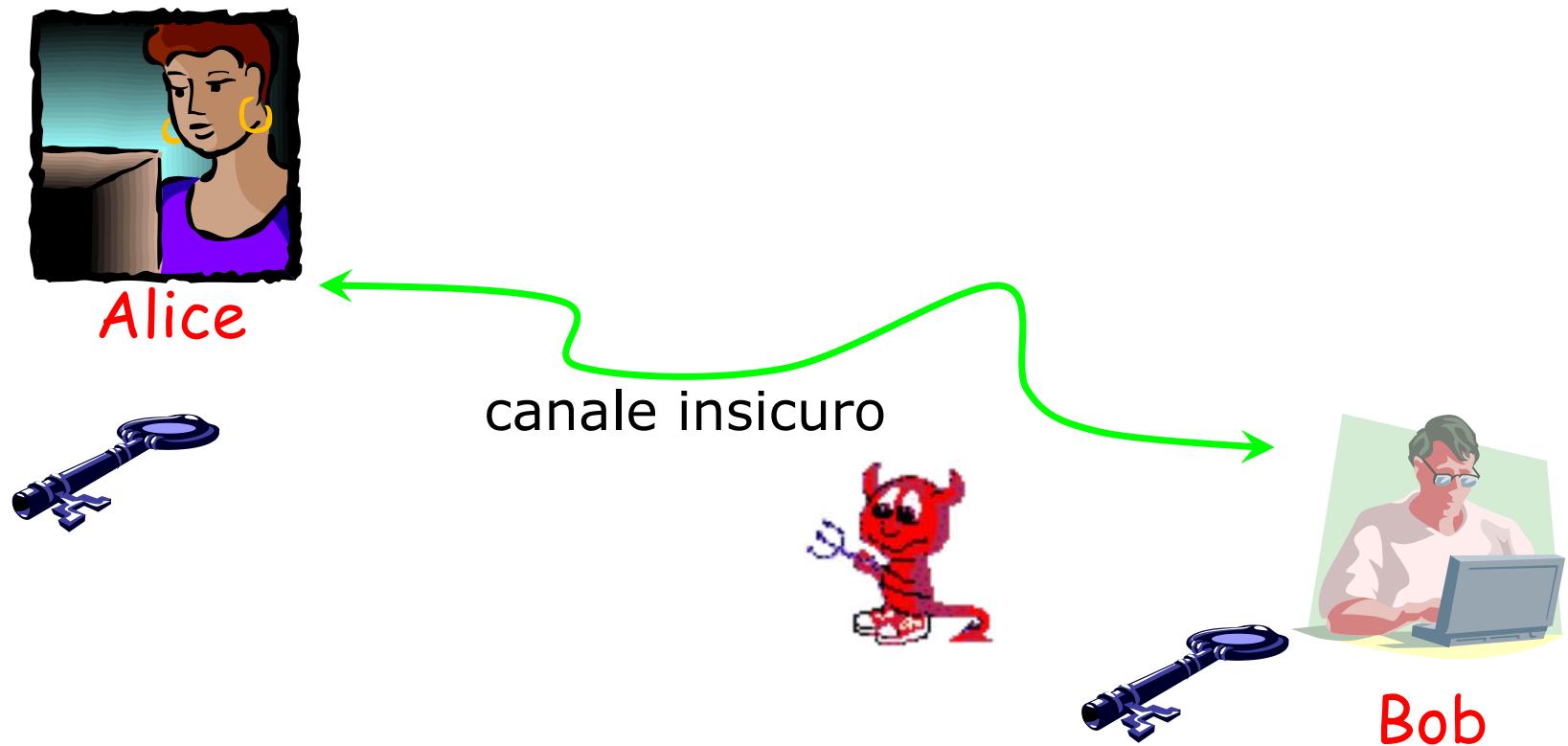


**Novembre 2020**

# Sommario

- RSA
- Descrizione
- Generazione dei parametri
- Sicurezza
  - Attacchi
  - Fattorizzazione
  - Lunghezza della chiave
- Uso in pratica

# Cifrari simmetrici



# Problemi

Come fanno Alice e Bob a condividere una chiave comune?



Uso di un canale privato

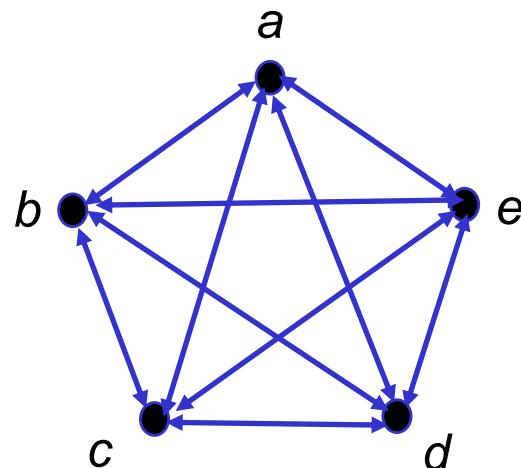
- un corriere fidato...
- un incontro faccia a faccia in un posto segreto...



# Gestione delle chiavi

In una rete con  $n$  utenti ogni coppia di utenti deve condividere una chiave

- Ogni utente deve memorizzare  $n-1$  chiavi
- Il numero totale delle chiavi segrete è'  $\binom{n}{2} = \frac{n(n-1)}{2}$



10 chiavi segrete: tra  
 $(a,b), (a,c), (a,d), (a,e), (b,c),$   
 $(b,d), (b,e), (c,d), (c,e), (d,e)$

# Gestione delle chiavi

In una rete con  $n$  utenti ogni coppia di utenti deve condividere una chiave

- Ogni utente deve memorizzare  $n-1$  chiavi
- Il numero totale delle chiavi segrete è  $\binom{n}{2} = \frac{n(n-1)}{2}$



L'aggiunta di un nuovo utente alla rete implica la distribuzione della chiave a tutti i precedenti utenti...

# Problemi

Come fanno Alice e Bob a condividere una chiave comune?

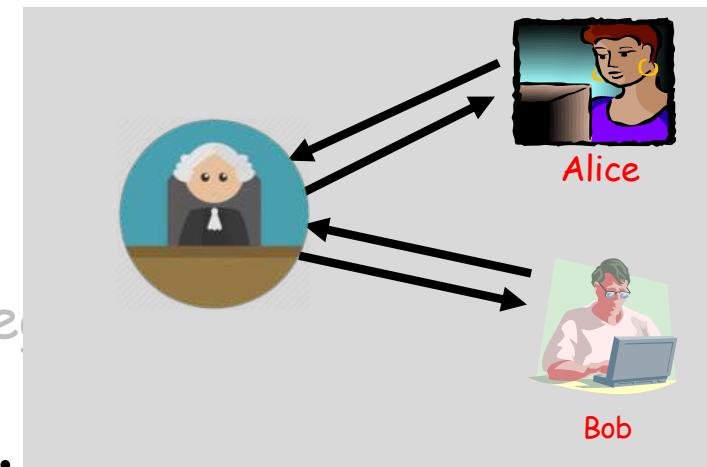


Uso di un canale privato

- un corriere fidato...
- un incontro faccia a faccia in un posto sicuro...

Uso di una terza parte fidata...

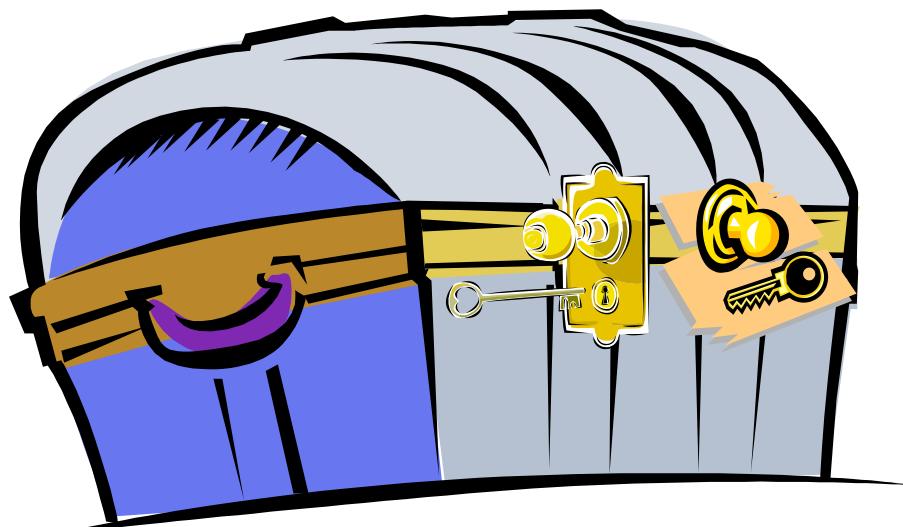
- che stabilisce la chiave di sessione e la invia ad entrambi in modo sicuro...



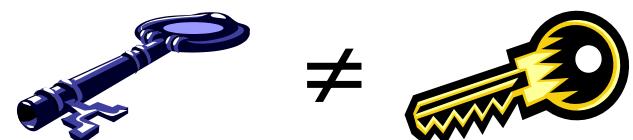
# Cifrari asimmetrici

Usano una cassaforte con due lucchetti

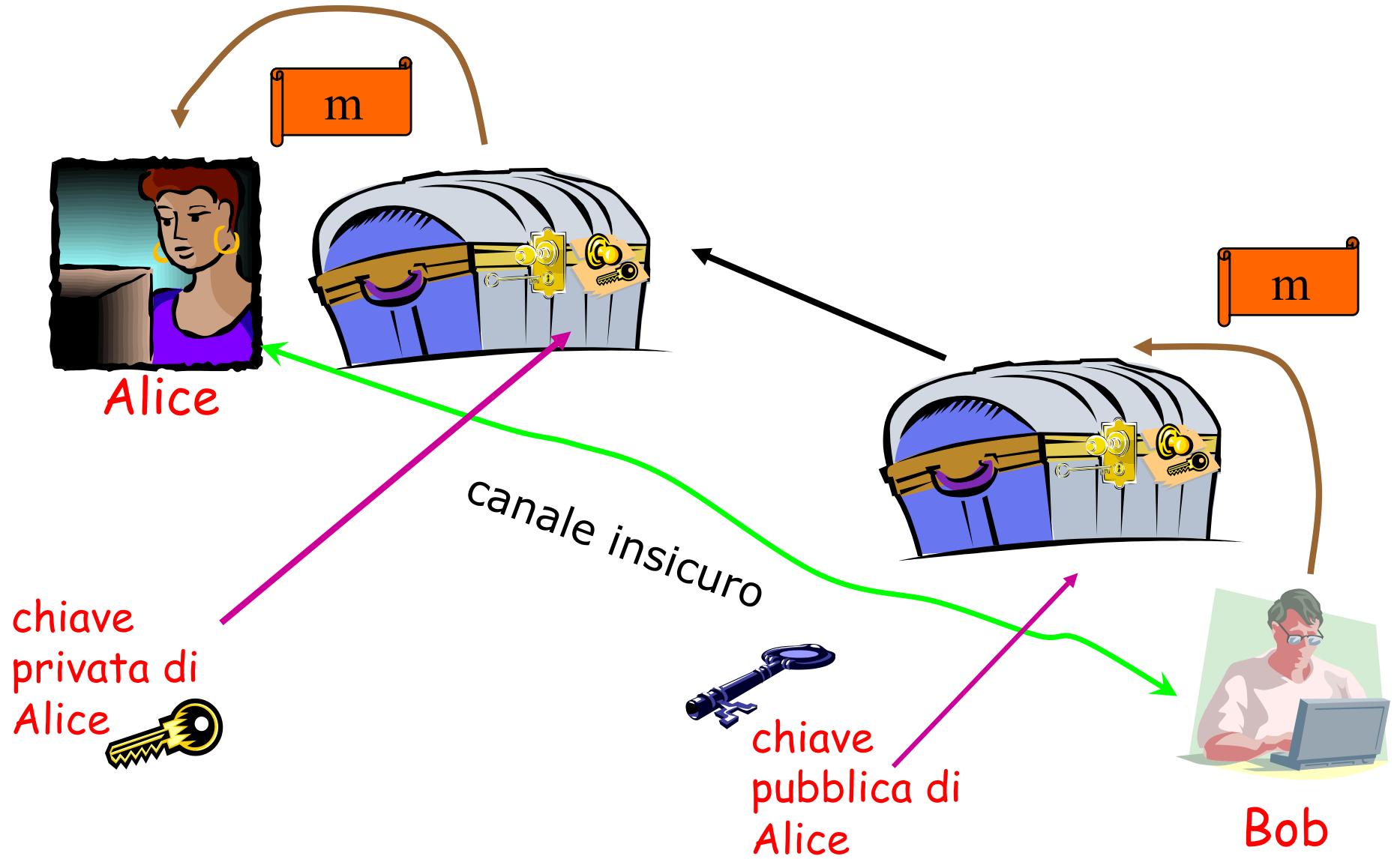
- Con una chiave (**pubblica**) chiudiamo la cassaforte
- Con l'altra chiave (**privata**) apriamo la cassaforte



Public key ≠ Private key



# Cifrari asimmetrici



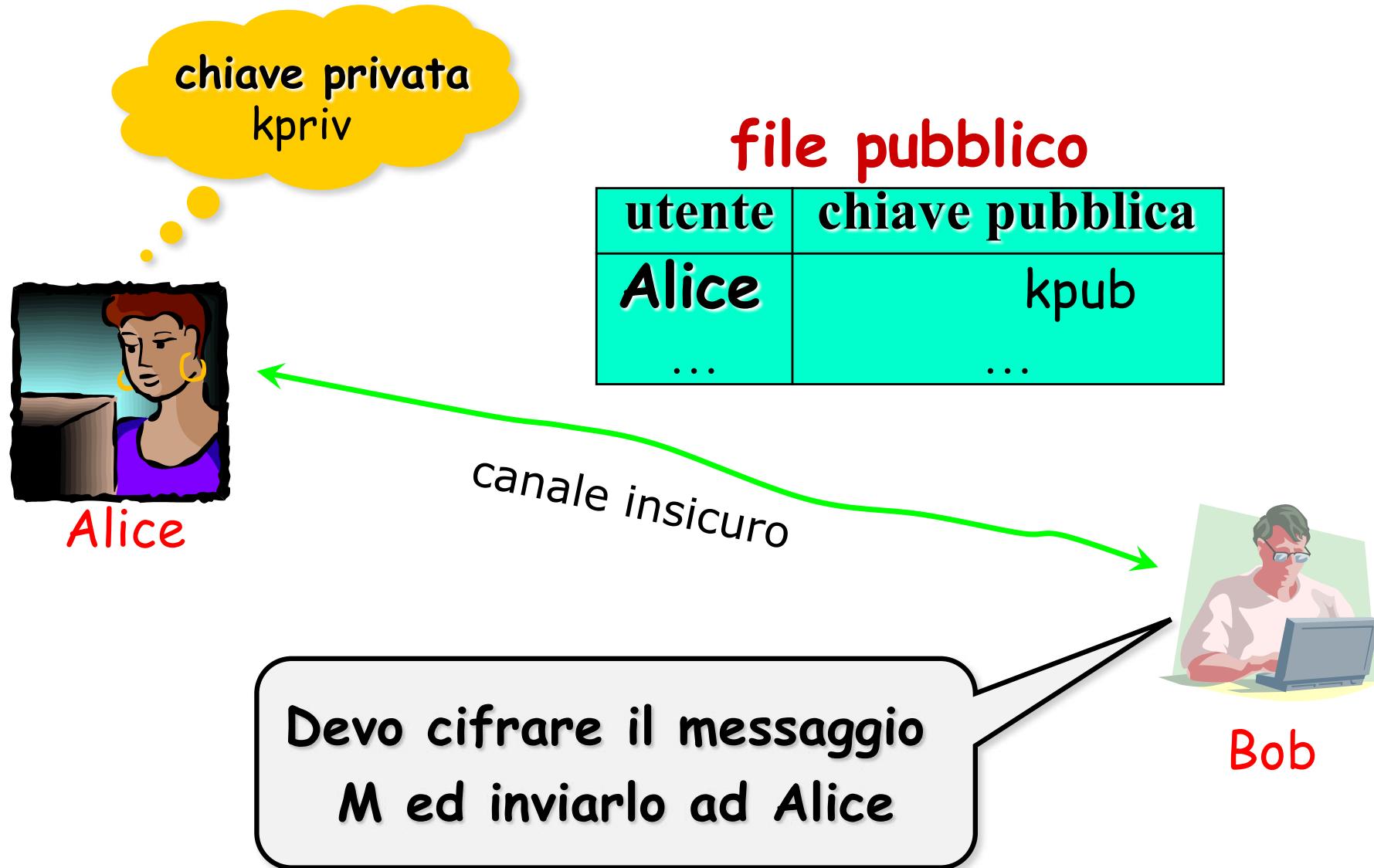
# Cifrari asimmetrici



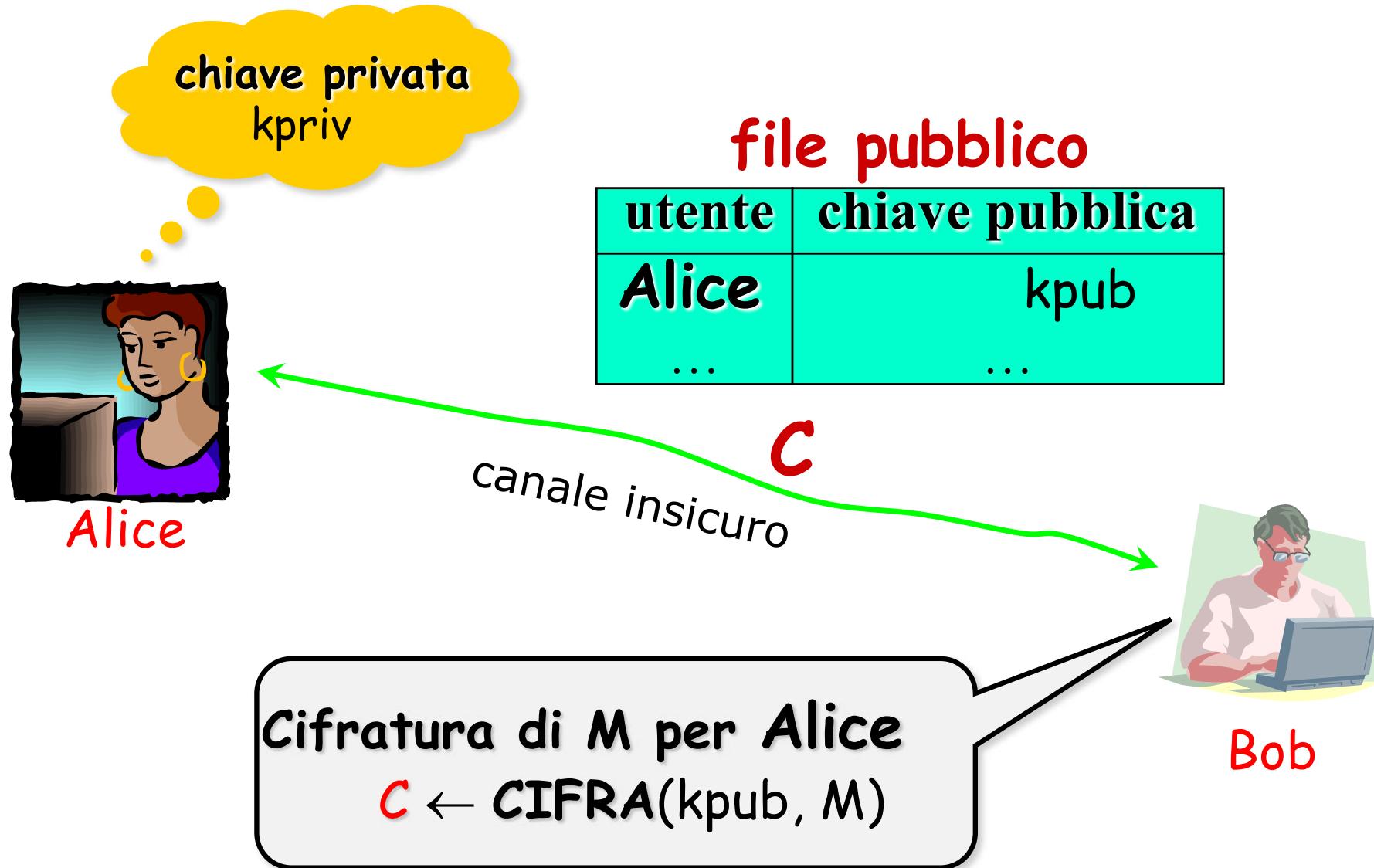
file pubblico

utente	chiave pubblica
Alice	kpub
...	...

# Cifratura



# Cifratura



# Decifratura

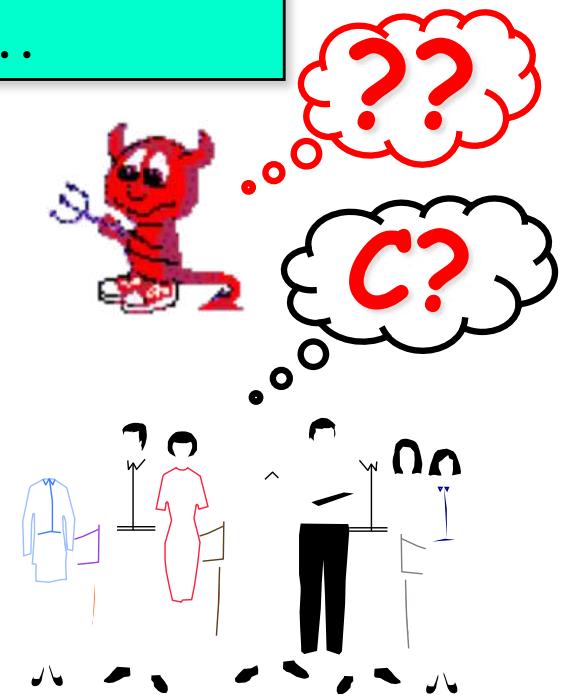
Devo decifrare il messaggio cifrato **C**



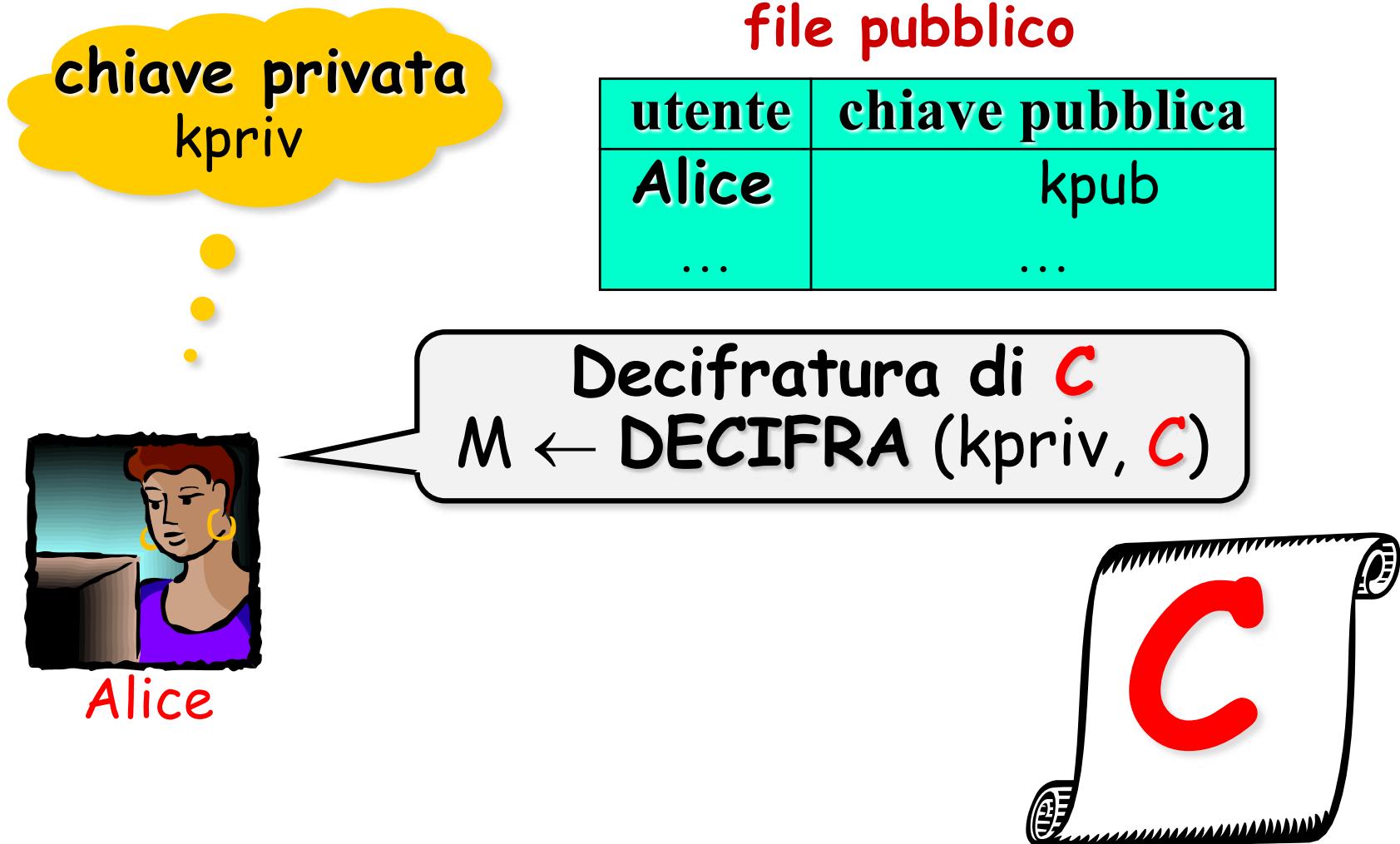
Alice

file pubblico

utente	chiave pubblica
Alice	kpub
...	...



# Decifratura



# Cifrari asimmetrici

- Chiunque può cifrare un messaggio per Alice
- Solo Alice può decifrare un messaggio cifrato per lei
- Non ci sono chiavi condivise tra gli utenti
  - Ogni utente genera da solo la propria coppia di chiavi (public key, private key) e rende pubblica la chiave pubblica
  - Ogni utente memorizza una sola chiave (privata)

# Cifrari asimmetrici

Come realizzarli?



# Funzioni one-way

"Facili" da calcolare e



"difficili" da invertire



# Funzioni one-way

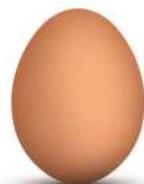
"Facili" da calcolare e



"difficili" da invertire



Esistono in natura, ma in Informatica?



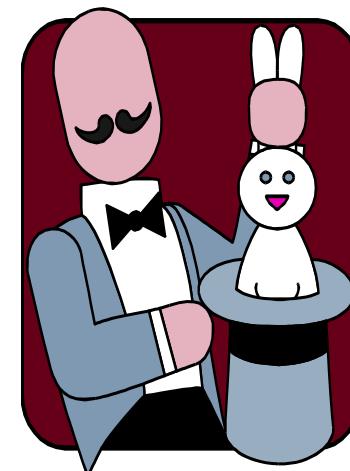
# Funzioni one-way trapdoor



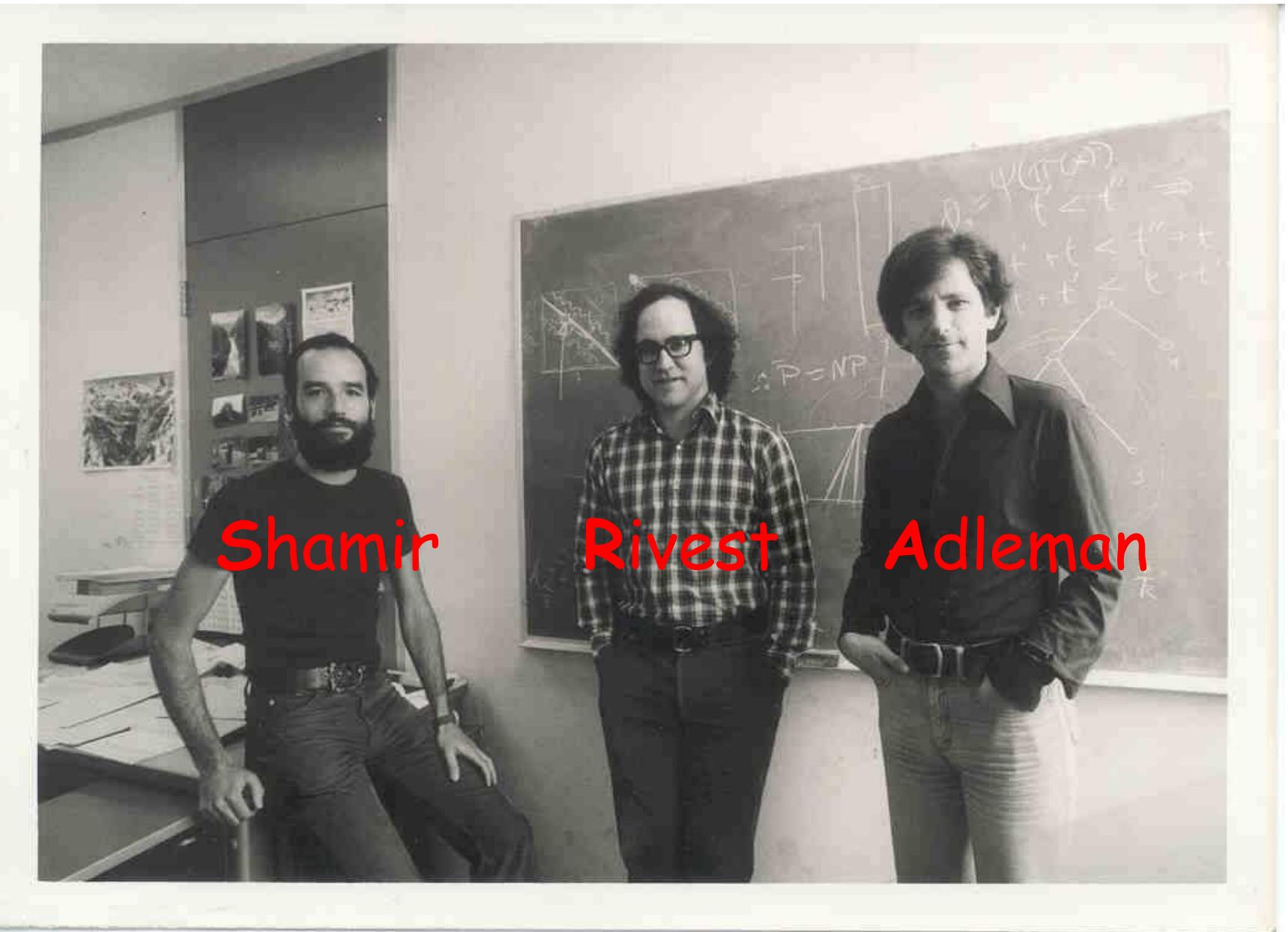
“Facili” da calcolare e  
“difficili” da invertire



... a meno che si conosca  
una “trapdoor”



# RSA [1978]

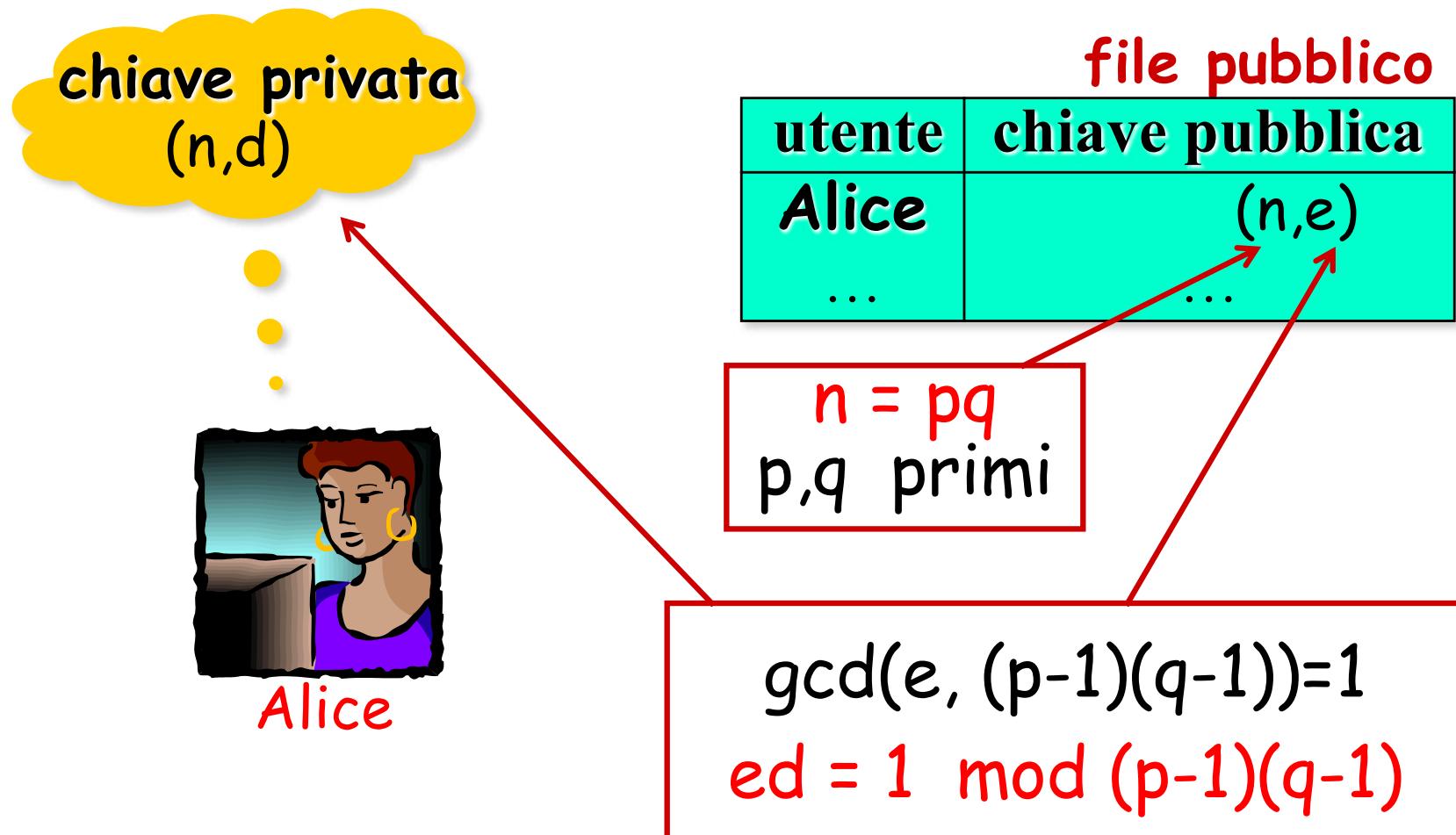


Shamir

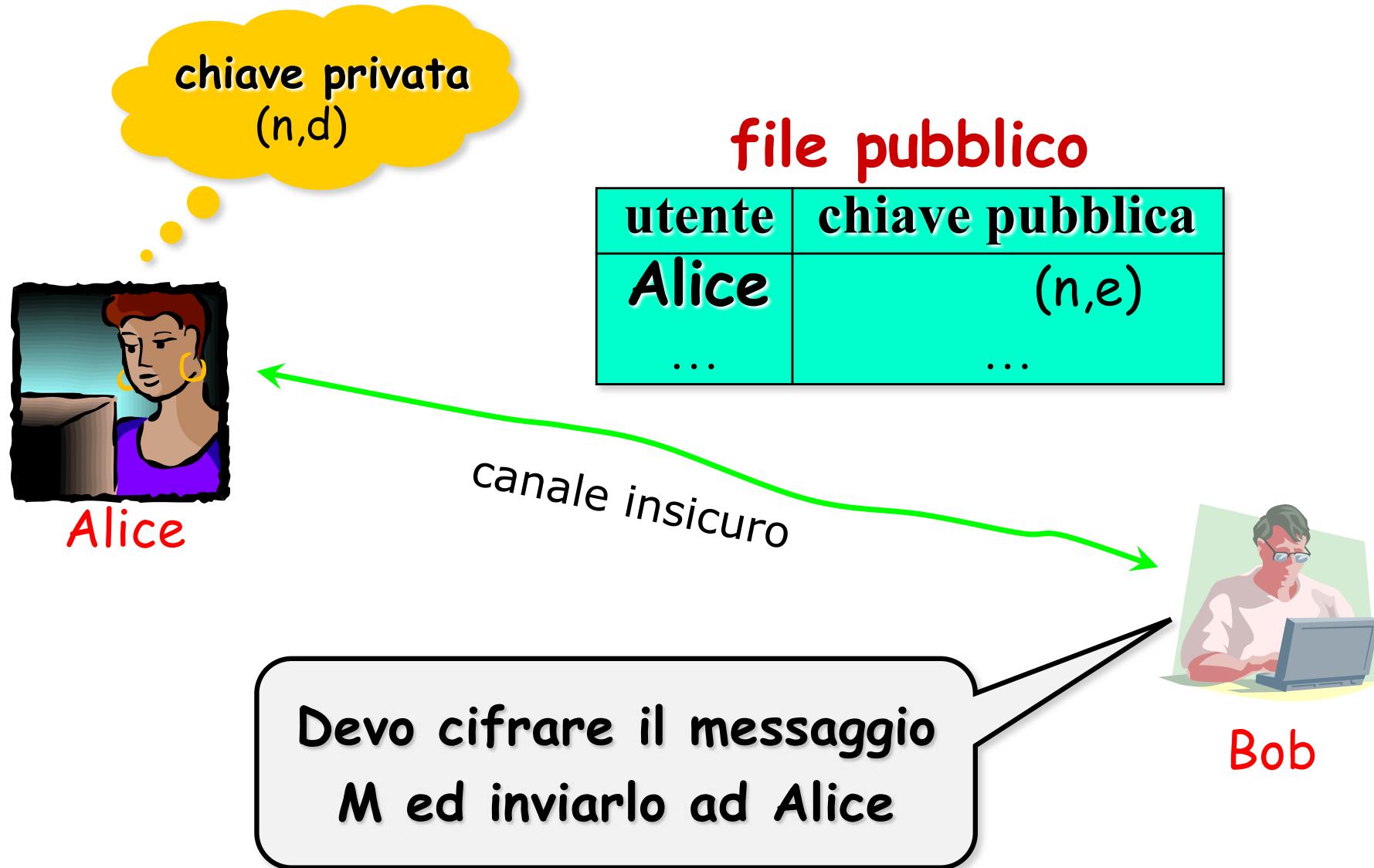
Rivest

Adleman

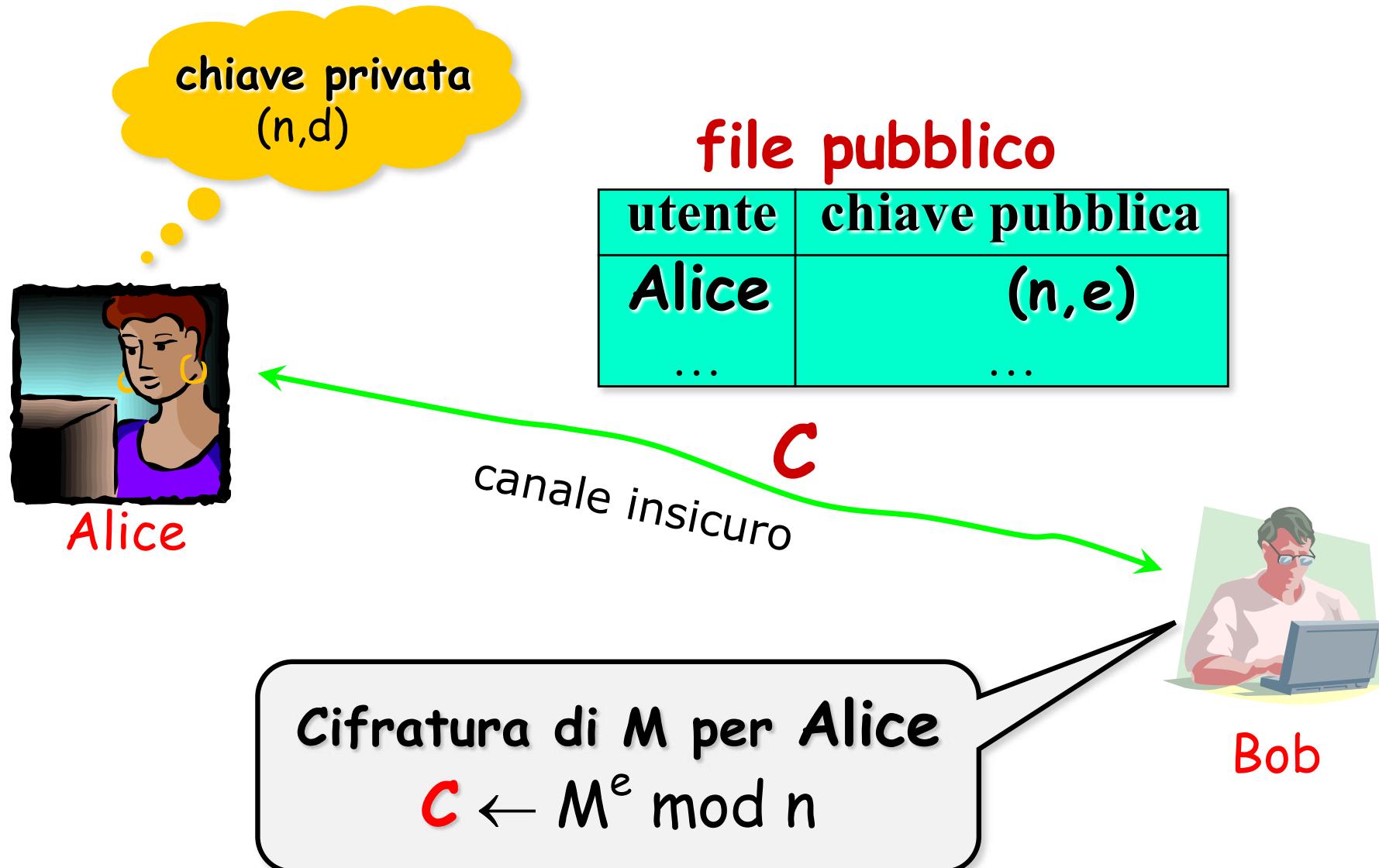
# Chiavi RSA



# Cifratura RSA



# Cifratura RSA



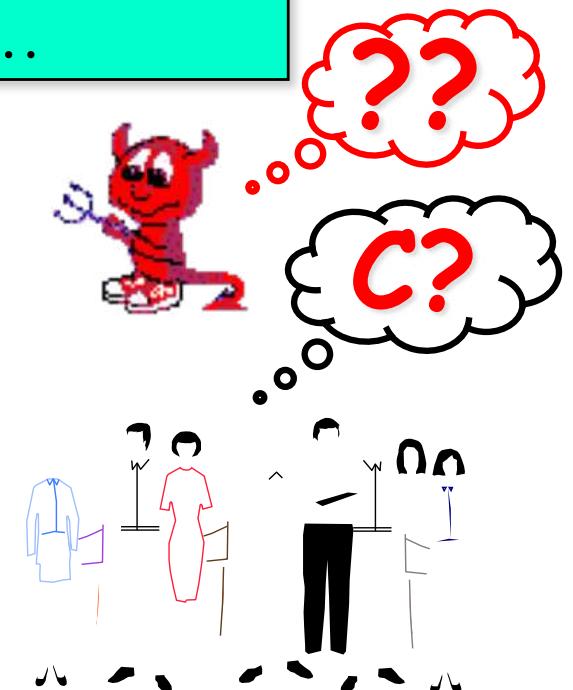
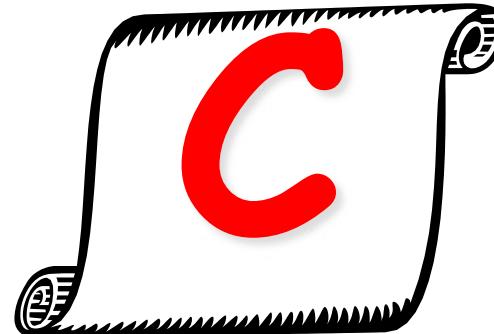
# Decifratura RSA

Devo decifrare il messaggio cifrato **C**

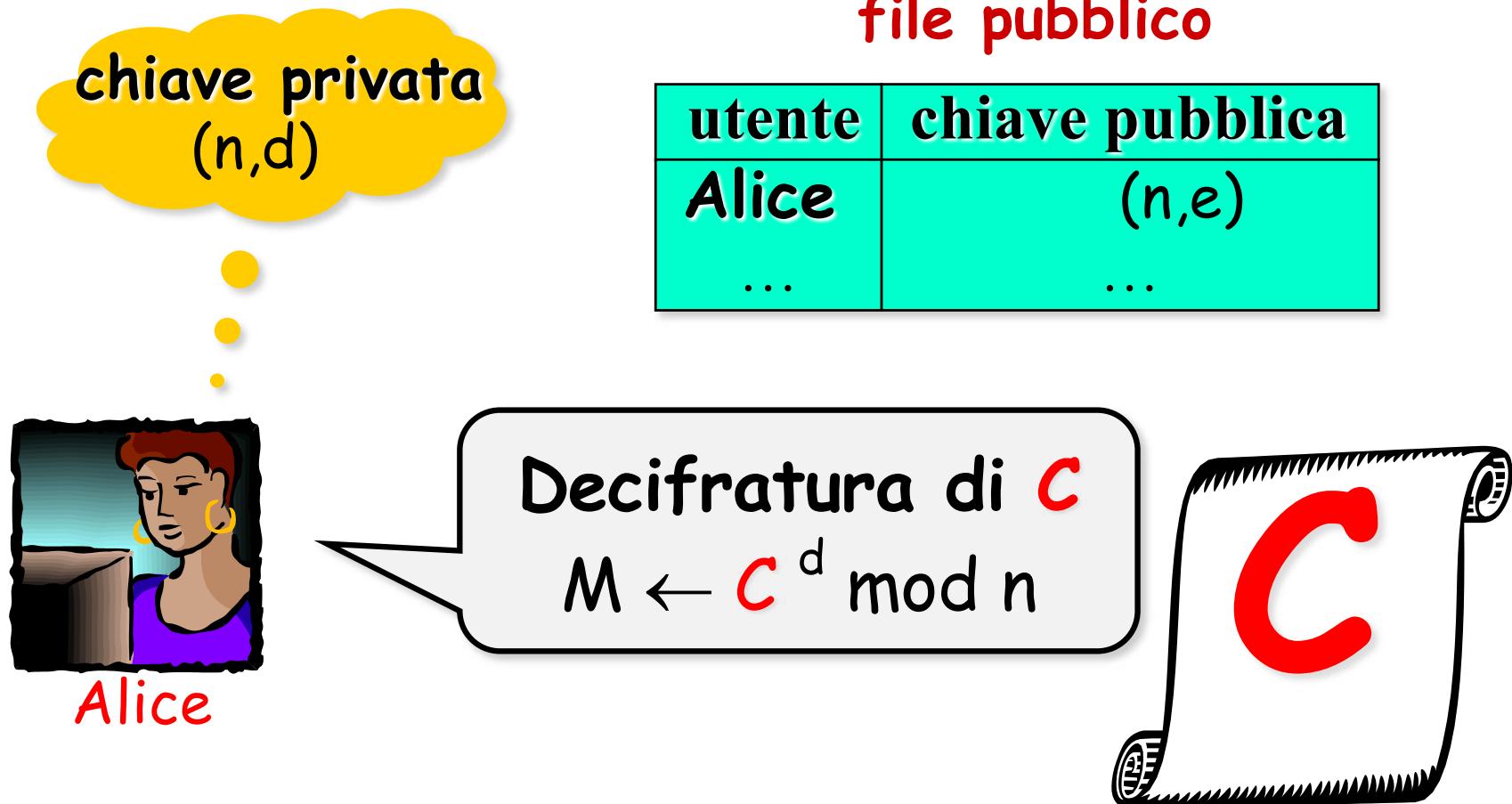


Alice

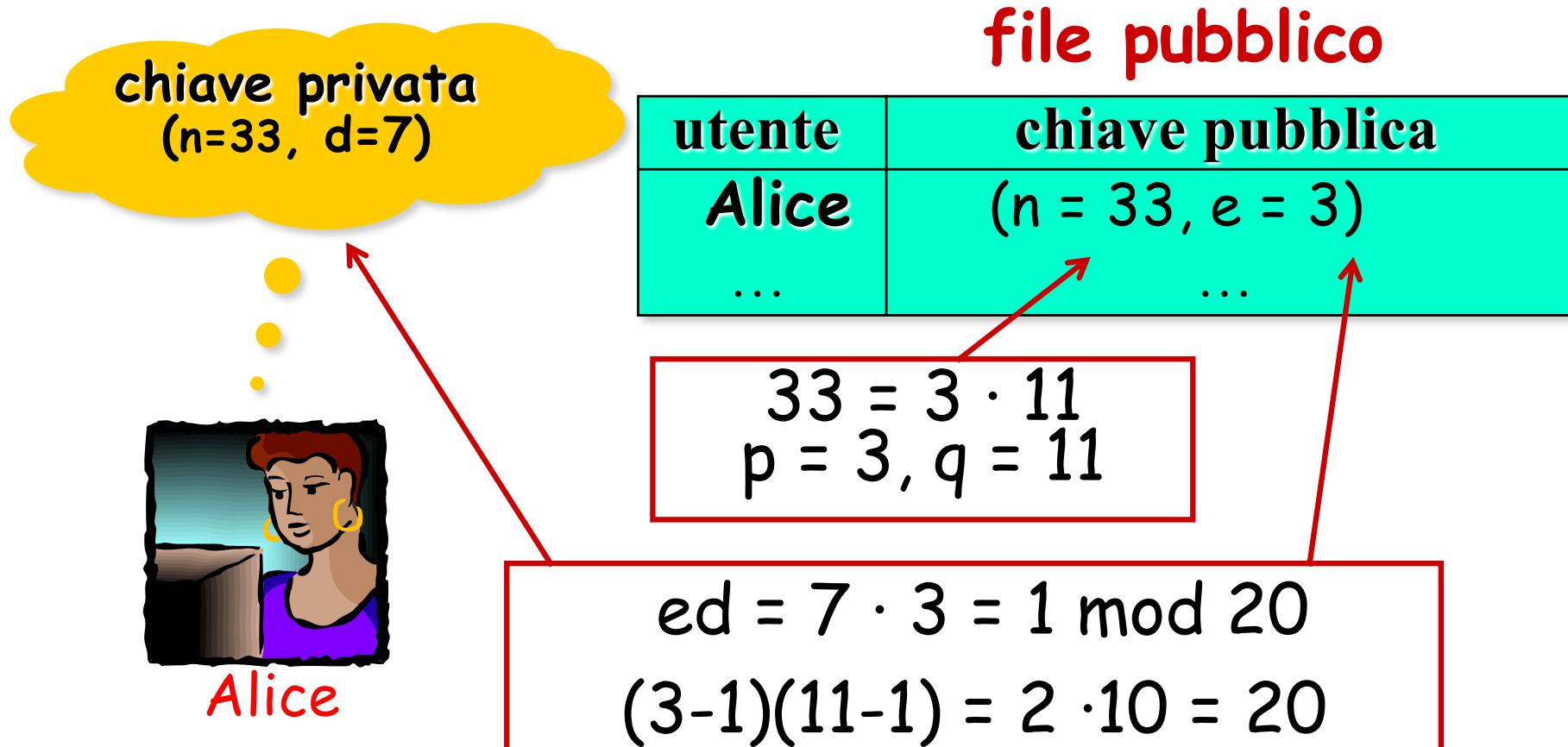
utente	chiave pubblica
Alice	(n,e)
...	...



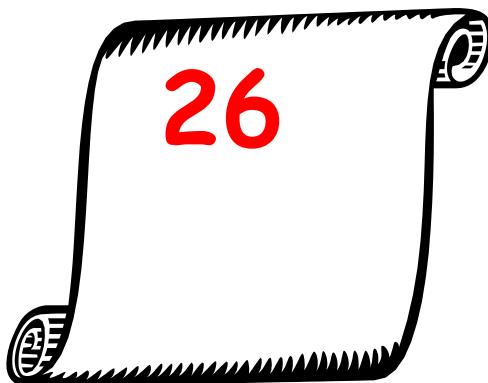
# Decifratura RSA



# “Piccolo” esempio: Chiavi RSA



# “Piccolo” esempio: Cifratura RSA



file pubblico

utente	chiave pubblica
Alice	(n = 33, e = 3)
...	...

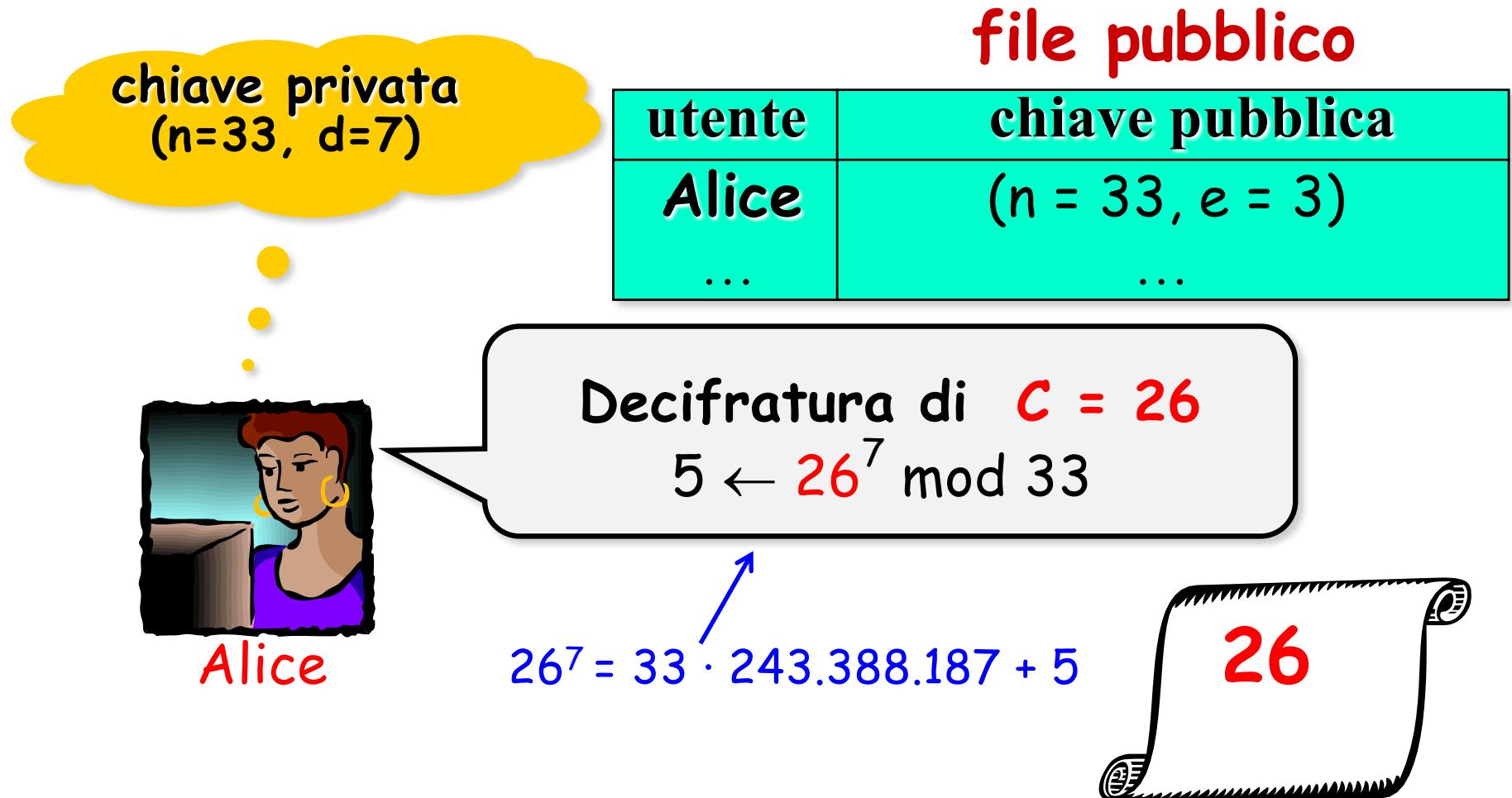
Cifratura di  $M = 5$  per Alice

$$26 \leftarrow 5^3 \bmod 33$$

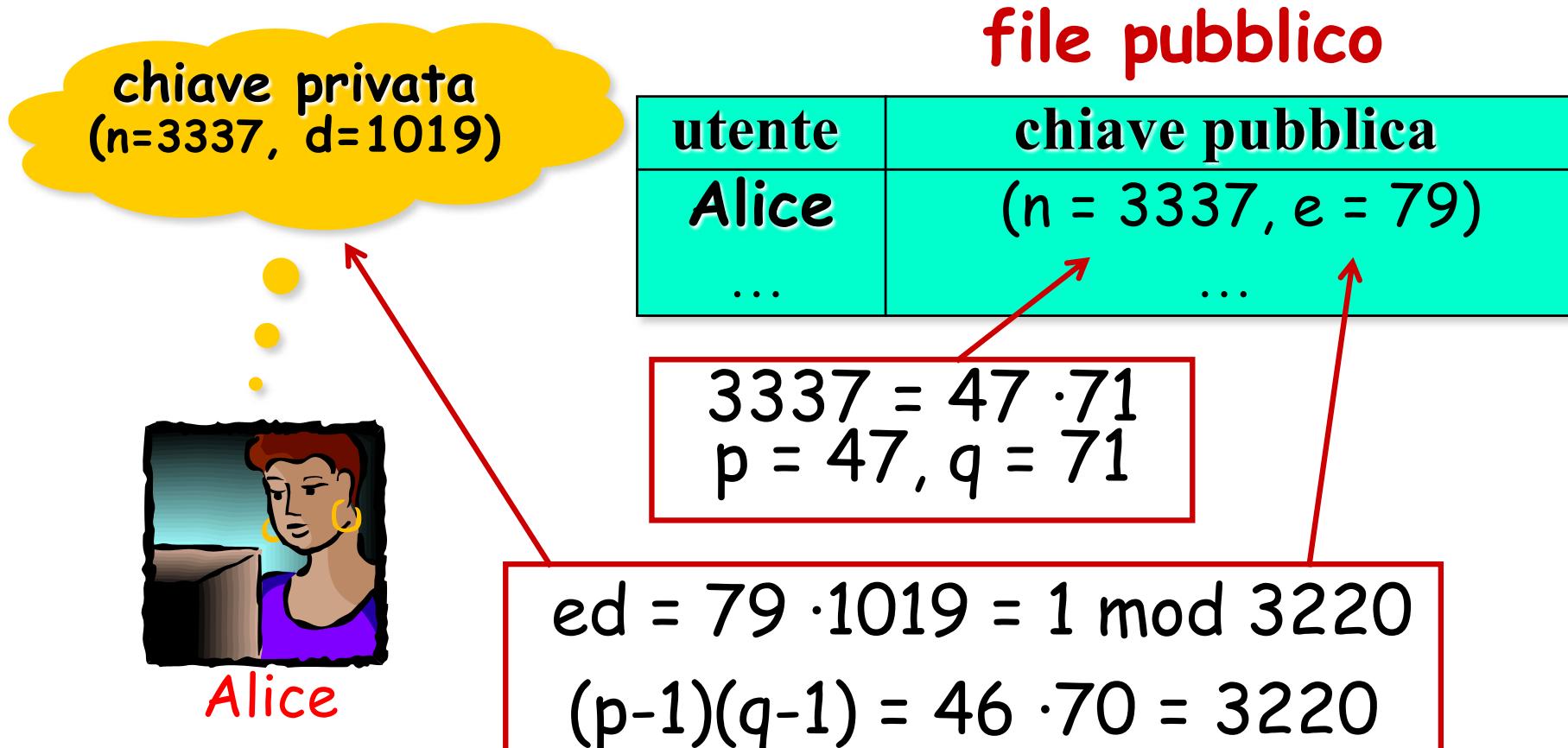


Bob

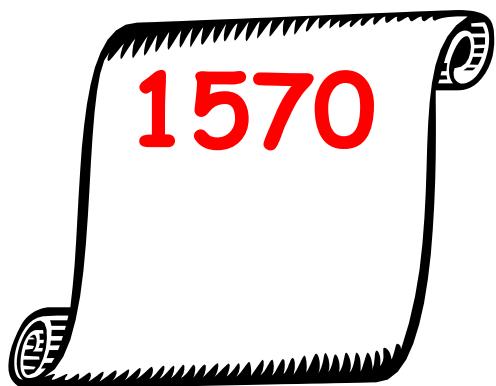
# “Piccolo” esempio: Decifratura RSA



# “Piccolo” esempio: Chiavi RSA



# “Piccolo” esempio: Cifratura RSA



file pubblico

utente	chiave pubblica
Alice	( $n = 3337$ , $e = 79$ )
...	...

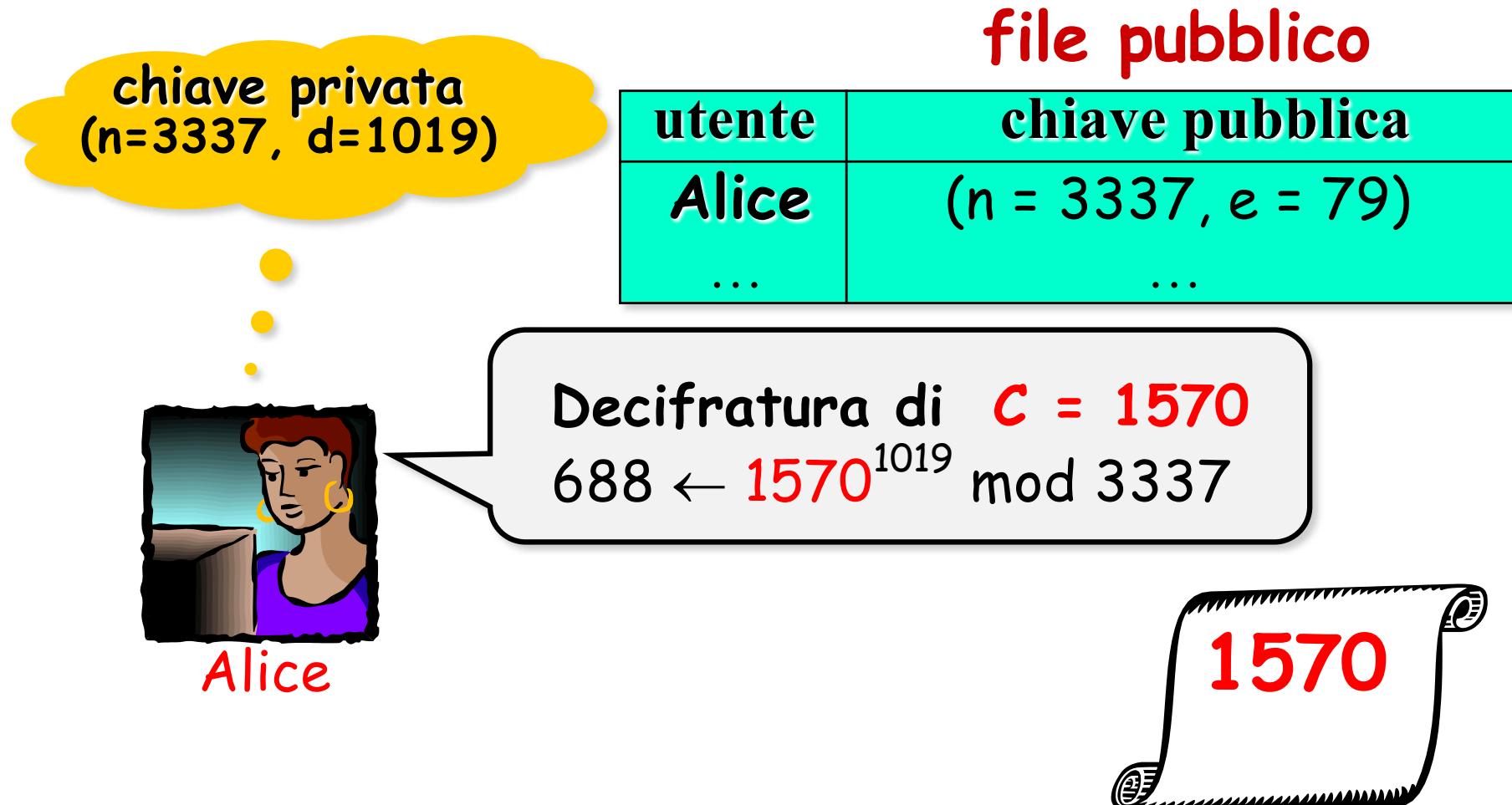
Cifratura di  $M = 688$  per Alice

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



Bob

# “Piccolo” esempio: Decifratura RSA



# esempio: RSA

Primi  $p$  e  $q$  di 512 bit

$p$

961303453135835045741915812806154279093098455949962158225831508796  
479404550564706384912571601803475031209866660649242019180878066742  
1096063354219926661209

$q$

120601919572314469182767942044508960015559250546370339360617983217  
314821484837646592153894532091752252732268301071206956046025138871  
45524969000359660045617

$n=pq$

115935041739676149688925098646158875237714573754541447754855261376  
147885408326350817276878815968325168468849300625485764111250162414  
552339182927162507656772727460097082714127730434960500556347274566  
628060099924037102991424472292215772798531727033839381334692684137  
327622000966676671831831088373420823444370953

# esempio: RSA

Primi p e q di 512 bit

(p-1)(q-1)

115935041739676149688925098646158875237714573754541447754855261376  
147885408326350817276878815968325168468849300625485764111250162414  
552339182927162507656751054233608492916752034482627988117554787657  
013923444405716989581728196098226361075467211864612171359107358640  
614008885170265377277264467341066243857664128

e

35535

d

580083028600377639360936612896779175946690620896509621804228661113  
805938528223587317062869100300217108590443384021707298690876006115  
306202524959884448047568240966247081485817130463240644077704833134  
010850947385295645071936774061197326557424237217617674620776371642  
0760033708533328853214470885955136670294831

# **esempio: cifratura RSA**

Messaggio THIS IS A TEST

# esempio: cifratura RSA

Messaggio T H I S     I S     A     T E S T  
19 07 08 18 26 08 18 26 00 26 19 04 18 19

(codifica da 00 a 25, con 26=“spazio”)

# esempio: cifratura RSA

Messaggio T H I S     I S     A     T E S T  
19 07 08 18 26 08 18 26 00 26 19 04 18 19

M

1907081826081826002619041819

$C=M^e$

475309123646226827206365550610545180942371796070491716523239243054  
452960613199328566617843418359114151197411252005682979794571736036  
101278218847892741566090480023507190715277185914975188465888632101  
148354103361657898467968386763733765777465625079280521148141844048  
14184430812773059004692874248559166462108656

# esempio: decifratura RSA

Messaggio THIS IS A TEST

M

1907081826081826002619041819

$C = M^e$

475309123646226827206365550610545180942371796070491716523239243054  
452960613199328566617843418359114151197411252005682979794571736036  
101278218847892741566090480023507190715277185914975188465888632101  
148354103361657898467968386763733765777465625079280521148141844048  
14184430812773059004692874248559166462108656

$C^d$

1907081826081826002619041819

# Esercizio

Costruire un piccolo esempio numerico per RSA

- Generazione chiavi
- Cifratura
- Decifratura

# Esercizio

Alice vorrebbe scegliere come chiave pubblica ( $n = 33$ ,  $e = 2$ )

- E' una buona scelta?
- Cifrare  $M=13$ 
  - Decifrare il valore ottenuto
  - Quanti testi in chiaro ci sono?

# Esercizio

Alice vorrebbe scegliere come chiave pubblica ( $n = 33$ ,  $e = 2$ )

- E' una buona scelta?
- Cifrare  $M=13$ 
  - Decifrare il valore ottenuto
  - Quanti testi in chiaro ci sono?

$M$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$M^2 \text{ mod } 33$	1	4	9	16	25	3	16	31	15	1	22	12	4	31	27	25
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	
25	27	31	4	12	22	1	15	31	16	3	25	16	9	4	1	

# Esercizio

Alice ha scelto come chiave pubblica  
 $(n = 33, e = 3)$

- Cifrare  $M=13$
- Decifrare il valore ottenuto
- Quanti testi in chiaro ci sono?

$M$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$M^3 \text{ mod } 33$	1	8	27	31	26	18	13	17	3	10	11	12	19	5	9	4
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	
29	24	28	14	21	22	23	30	16	20	15	7	2	6	25	32	

# Correttezza decifratura RSA

$$C^d \bmod n = (M^e)^d \bmod n$$

$$= M^{ed} \bmod n$$

$ed = 1 \bmod (p-1)(q-1)$

$$= M^{1+k(p-1)(q-1)} \bmod n$$

$$= M \cdot (M^{(p-1)(q-1)})^k$$

$$= M \bmod n$$

Teorema di Eulero  
 $M \in \mathbb{Z}_n^* \Rightarrow M^{(p-1)(q-1)} = 1 \bmod n$

$$= M$$

poichè  $0 \leq M < n$

# Correttezza decifratura RSA

$$C^d \bmod n = (M^e)^d \bmod n$$

$$= M^{ed} \bmod n$$

$$ed = 1 \bmod (p-1)(q-1)$$

$$= M^{1+k(p-1)(q-1)} \bmod n$$

$$= M \cdot (M^{(p-1)(q-1)})^k$$

$$= M \bmod n$$

Teorema di Eulero  
 $M \in \mathbb{Z}_n^* \Rightarrow M^{(p-1)(q-1)} = 1 \bmod n$

$$= M$$

Per  $M \in \mathbb{Z}_n / \mathbb{Z}_n^*$  usa  
il **teorema cinese del resto**

poichè  $0 \leq M < n$

# Efficienza delle computazioni

RSA utilizza le seguenti computazioni

- Generazione numeri primi p e q
- Generazione  $e, d$   $\left\{ \begin{array}{l} \text{generazione di } e \\ d \leftarrow e^{-1} \bmod (p-1)(q-1) \end{array} \right.$
- Elevazione a potenza modulare
  - Per cifratura e decifratura



# Generazione chiavi

1. Input  $L$  (lunghezza modulo)
2. Genera 2 primi di lunghezza  $L/2$
3.  $n \leftarrow p \cdot q$
4. Scegli a caso  $e$
5. If  $\gcd(e, (p-1)(q-1)) = 1$   
    then  $d \leftarrow e^{-1} \bmod (p-1)(q-1)$   
    else goto 4.

# Scelta esponente pubblico

Minimizzare operazioni per elevazione a potenza

- $e \leftarrow 3$
- $e \leftarrow 2^{16}+1$ 
  - decimale 65.537
  - binario 1000000000000001
  - Più comune: calcolo di  $M^e$  richiede solo 16 quadrati ed una moltiplicazione

# Generazione chiavi (comunemente usata in pratica)

1. Input  $L$  (lunghezza modulo)
2.  $e \leftarrow 3$  oppure  $e \leftarrow 2^{16}+1$
3. Genera 2 primi di lunghezza  $L/2$
4.  $n \leftarrow p \cdot q$
5. If  $\gcd(e, (p-1)(q-1)) = 1$   
    then  $d \leftarrow e^{-1} \bmod (p-1)(q-1)$   
    else goto 3.

# Prestazioni implementazioni

AMD Opteron 8354 2.2GHz, Linux, Crypto++ 5.6.0 Benchmarks

- routine assembly language per aritmetica su interi

	bit chiave	cifratura	decifratura
RSA	1024	0,08	1,46
RSA	2048	0,16	6,08

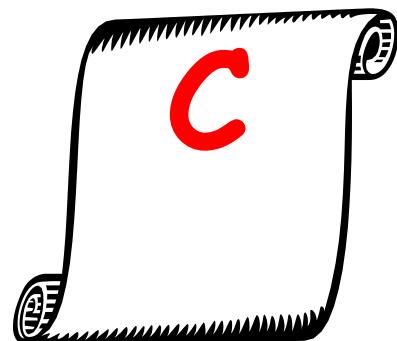
millisecondi/operazione

esponente  
pubblico RSA: 17

<http://www.cryptopp.com/benchmarks.html> (31 marzo 2009)

# Sicurezza di RSA

Sicurezza della generazione delle chiavi



Sicurezza della cifratura

# Sicurezza generazione chiavi di RSA



Conoscendo la chiave pubblica **(n,e)**



vuole calcolare la chiave privata

$$d = e^{-1} \bmod (p-1)(q-1)$$

# Attacco 1: fattorizzare n

Se  potesse fattorizzare n, saprebbe  
computare d

# Attacco 1: fattorizzare n

Se  potesse fattorizzare n, saprebbe  
computare d

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

# Attacco 2: computare $\varphi(n)$

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

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

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

# Attacco 2: computare $\varphi(n)$

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

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

$$\begin{aligned} &\text{sostituendo } p = n/q \\ &p^2 - (n - \varphi(n) + 1)p + n = 0 \end{aligned}$$

$$\frac{(n - \varphi + 1) \pm \sqrt{(n - \varphi(n) + 1)^2 - 4 \cdot n}}{2}$$

Due soluzioni:  $p, q$

# Attacco 2: computare $\varphi(n)$

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

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

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: 9.539 e 8.887

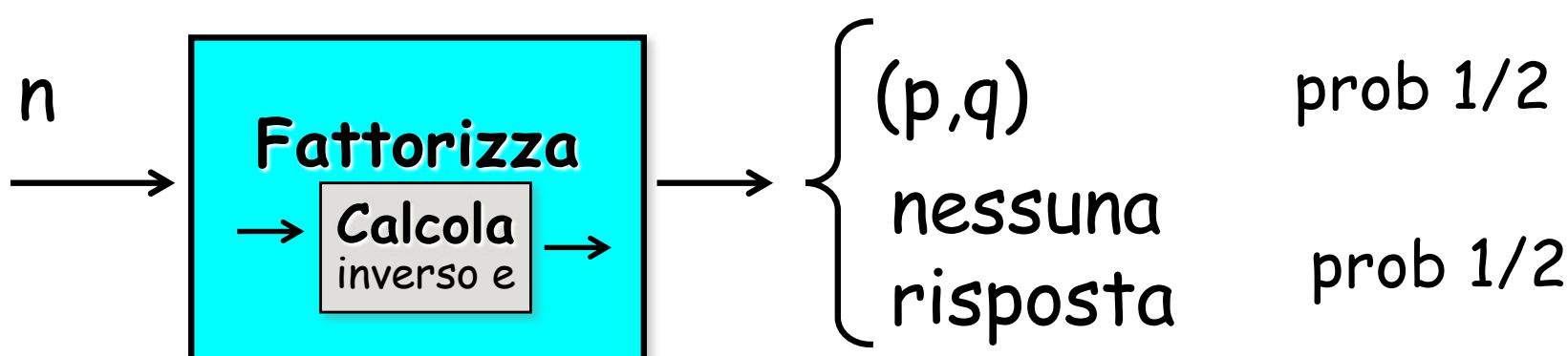
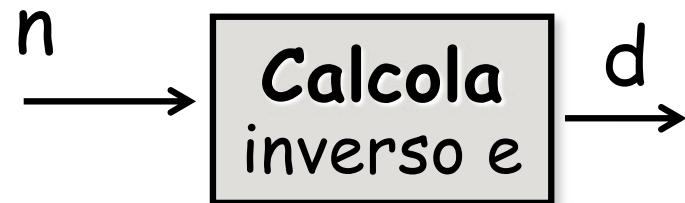
# Attacco 3: computare d



Se  potesse computare d saprebbe  
fattorizzare n

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

# Algoritmo Las Vegas per fattorizzare



# Sicurezza generazione chiavi di RSA



Fattorizza  $n \rightarrow$  Computa  $d$

Computa  $d \rightarrow$  Fattorizza  $n$

Computare  $d$  è equivalente a fattorizzare  $n$

# Fattorizzazione

- Dato  $n$ , calcolare due primi  $p, q > 1$  tali che  $n = pq$
- Per valori grandi di  $n$  è un problema ritenuto **computazionalmente difficile**
- Complessità di tempo **sub-esponenziale** in media
  - Running time  $O(2^{o(k)})$ , dove  $k$  è la taglia dell'input
  - $f(n) = o(g(n))$  se  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$

# Fattorizzazione

## Teorema fondamentale dell'aritmetica

Ogni intero composto  $n > 1$  può essere scritto in modo unico come prodotto di potenze di primi

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

$p_i$  primo,  $e_i$  intero positivo

Prima dimostrazione:

Gauss, *Disquisitiones Arithmeticae*, 1798  
(in latino, quando aveva 21 anni)

# 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  $n$

Esempio:

- $n=77$ , numeri primi in  $[2, \sqrt{77}] = 2, 3, 5, 7$   $77=7 \cdot 11$
- $n=143$ , numeri primi in  $[2, \sqrt{143}] = 2, 3, 5, 7, 11$   $143=11 \cdot 13$

# 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  $\Theta(\sqrt{n}) = \Theta(2^{\frac{1}{2} \log_2 n})$

(esponenziale nella lunghezza dell'input)

Se  $n$  ha 2048 bit allora  $\sqrt{n} \approx 2^{1024}$



# 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**:  $L_n[1/2, 1]$   $\left(\frac{64}{9}\right)^{1/3} = 1.922999$
- **General Number Field Sieve**:  $L_n[1/3, 1.923]$



# Fattorizzazione: “idea” complessità algoritmi

Funzione  $n$ :  
 $\log n$

Funzione  $\sqrt{n}$ :  
 $0.5 \log n$

Quadratic sieve:  $\log(e^{(\ln n)^{1/2}}(\ln \ln n)^{1/2})$

General number field sieve:  
 $\log(e^{1.923(\ln n)^{1/3}}(\ln \ln n)^{2/3})$

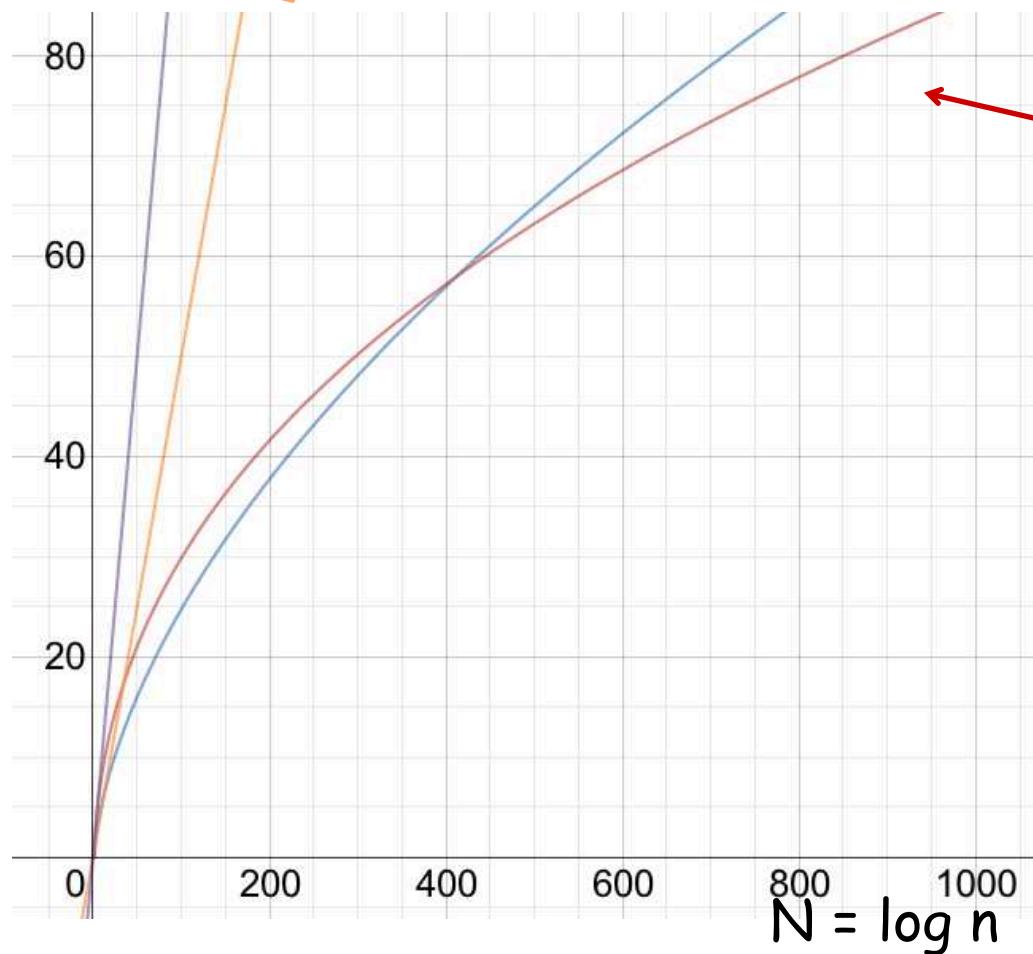
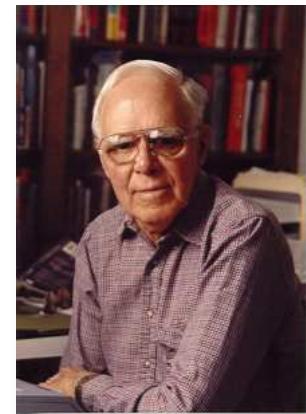


Grafico ottenuto  
sostituendo  $n=2^N$

# Fattorizzazione: sfide



Martin Gardner, *Mathematical Games*, Scientific American, 1977  
In questo lavoro gli inventori di RSA pubblicarono questa sfida

**Decifrare:**

9686 9613 7546 2206 1477 1409 2225 4355 8829 0575 9991 1245 7431  
9874 6951 2093 0816 2982 2514 5708 3569 3147 6622 8839 8962 8013  
3919 9055 1829 9451 5781 5154

**Chiave RSA:**

*n* 114 381 625 757 888 867 669 235 779 976 146 612 010 218 296 721 242  
362 562 561 842 935 706 935 245 733 897 830 597 123 513 958 705 058  
989 075 147 599 290 026 879 543 541

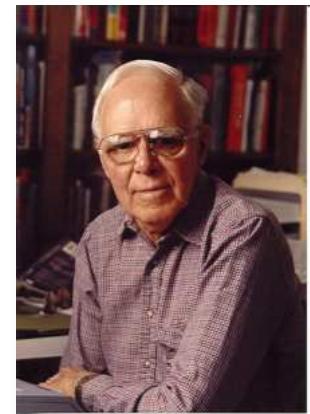
*e* 9007

Chiave di 426 bit, premio 100 \$

Stimarono il tempo richiesto: 40 quadrilioni di anni

# Fattorizzazione: sfide

Martin Gardner, *Mathematical Games*, Scientific American, 1977  
In questo lavoro gli inventori di RSA pubblicarono questa sfida



- Nel 1994: task force di Internet (1.600 pc, 600 volontari) ha reclamato il premio dopo 9 mesi di lavoro
- Testo in chiaro:  
"The magic words are squeamish ossifrage"

# Fattorizzazione: sfide

## RSA factoring challenge

- Iniziata nel 1991, per incoraggiare la ricerca
- Lista di prodotti di 2 primi
- Generati da un computer
  - senza alcun tipo di connessione
  - hard disk distrutto dopo generazione
  - ... non ci sono tracce dei fattori primi
- Prima RSA-YYY in decimale poi RSA-XXXX in bit
- Dichiarata conclusa nel 2007
- Premi in dollari, poi annullati nel 2007
- <http://www.rsa.com/rsalabs/node.asp?id=2092>

# RSA factoring challenge

	Cifre decimali	Cifre binarie	Data fattorizzazione
RSA-100	100	330	Apr 1991
RSA-110	110	364	Apr 1992
RSA-120	120	397	Giu 1993
RSA-129	129	426	Apr 1994
RSA-130	130	430	Apr 1996
RSA-140	140	463	Feb 1999
RSA-150	150	496	Apr 2004
RSA-155	155	512	Ago 1999
RSA-160	160	530	Apr 2003
RSA-170	170	563	Dic 2009
RSA-576	174	576	Dic 2003
RSA-180	180	596	Mag 2010
RSA-640	193	640	Nov 2005
RSA-200	200	663	Mag 2005
RSA-768	232	768	Dic 2009
RSA-240	240	795	Dic 2019
RSA-250	250	829	Feb 2020

RSA-129 (Scientific American)  
1600 computer per 8 mesi  
Premio \$100

RSA-576  
Premio \$10.000

RSA-640  
Premio \$20.000

# RSA-200

- 200 cifre decimali
- Fattorizzato 9 maggio 2005 da F. Bahr, M. Boehm, J. Franke, e T. Kleinjung
- Tempo equivalente al lavoro di 75 anni di un singolo computer con processore 2.2 GHz AMD Opteron e 2 GB RAM

RSA-200 =

2799783391122132787082946763872260162107044678695542853756000992932  
61284001076093456710529553608560618223519109513657886371059544820065  
76775098580557613579098734950144178863178946295187237869221823983

p 353246193440277012127260497819846436867119740019762  
5023649303468776121253679423200058547956528088349

q 7925869954478333033347085841480059687737975857364  
219960734330341455767872818152135381409304740185467

# RSA-768

- 232 cifre decimali, ovvero 768 bit
- Fattorizzato 12 dicembre 2009 da Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman te Riele, Andrey Timofeev, e Paul Zimmermann
- Tempo equivalente al lavoro di 2.000 anni di un singolo computer con processore 2.2 GHz AMD Opteron e 2 GB RAM

RSA-768 =

123018668453011775513049495838496272077285356959533479219732245215  
17264005072636575187452021997864693899564749427740638459251925573  
2630345373154826850791702612214291346167042921431160222124047927473  
7794080665351419597459856902143413

p      33478071698956898786044169848212690817704794983713768568912  
      431388982883793878002287614711652531743087737814467999489

q      367460436667995904282446337996279526322791581643430876426  
      76032283815739666511279233373417143396810270092798736308917

# RSA-240

- 240 cifre decimali, ovvero 795 bit
- Fattorizzato 2 dicembre 2019 da F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé and P. Zimmermann
- Calcolato anche un logaritmo discreto di 768 bit
- Per entrambi, tempo totale equivalente al lavoro di 4.000 anni di un singolo computer con processore 2.1GHz Intel Xeon Gold 6130 CPUs as a reference
- Number Field Sieve algorithm, usando open-source CADO-NFS software

RSA-240 =

124620366781718784065835044608106590434820374651678805754818788883  
28966680118821085503603957027250874750986476843845862105486553797  
025393057189121768431828636284694840530161441643046806687569941524  
6993185704183030512549594371372159029236099

p 5094359522858399145550510235808437141326483820241114731866602965218  
21206469746700620316443478873837606252372049619334517

q 244624208838318150567813139024002896653802092578931401452041221336  
558477095178155258218897735030590669041302045908071447

# RSA-250

- 250 cifre decimali, ovvero 829 bit
- Fattorizzato 28 febbraio 2020 da F. Boudot, P. Gaudry, A. Guillevic, N. Héninger, E. Thomé and P. Zimmermann
- Tempo equivalente al lavoro di 2.700 anni di un singolo computer con processore 2.1GHz Intel Xeon Gold 6130 CPUs as a reference
- Decine di migliaia di macchine, completata in pochi mesi
- Number Field Sieve algorithm, usando open-source CADO-NFS software

RSA-250 =

214032465024074496126442307283933356300861471514475501779775492088  
1418023447140136643345519095804679610992851872470914587687396261921  
557363047454770520805119056493106687691590019759405693457452230589  
325976697471681738069364894699871578494975937497937

p 641352894770715802787901901705773890848250147429434472081168596320  
24532344630238623598752668347708737661925585694639798853367

q 33372027594978156556226010605355114227940760344767554666784520987  
023841729210037080257448673296881877565718986258036932062711

# CADO-NFS

- Crible Algébrique: Distribution, Optimisation - Number Field Sieve
- Implementazione in C/C++ dell'algoritmo Number Field Sieve (NFS) per fattorizzare interi e calcolare logaritmi discreti in campi finiti
- <http://cado-nfs.gforge.inria.fr/>

# RSA factoring challenge

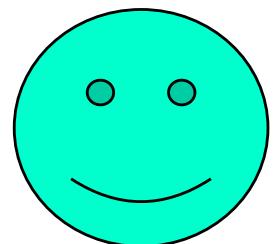
	Cifre decimali	Cifre binarie	Premio (ritirato)
RSA-260	260	862	-
RSA-270	270	895	-
RSA-896	270	896	\$75.000
...	...	...	...
RSA-309	309	1024	-
RSA-1024	309	1024	\$100.000
...	...	...	...
RSA-460	463	1536	-
RSA-1536	463	1536	\$150.000
...	...	...	...
RSA-490	490	1626	-
RSA-500	500	1659	-
RSA-2048	617	2048	\$200.000

- Non ancora fattorizzati
- Premi annullati



# 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      (RSA Lab., Cryptobytes, 1995)
  - 768 bit per uso personale
  - 1024 bit per le aziende
  - 2048 per chiavi "importanti"  
-ad esempio **Autorità di Certificazione**



# Che modulo scegliere?

Protection Lifetime of Data	Present – 2010	Present – 2030	Present – 2031 and Beyond
<b>Minimum symmetric security level</b>	80 bits	112 bits	128 bits
<b>Minimum RSA key size</b>	1024 bits	2048 bits	3072 bits

Table 1. Recommended minimum symmetric security levels and RSA key sizes based on protection lifetime.

- Maggio 2003
- <http://www.rsa.com/rsalabs/node.asp?id=2004>

# Che parametri scegliere?

Bisogna tener conto dell'  
expected security life

Table 4: Recommended algorithms and minimum key sizes

Algorithm security lifetimes	Symmetric key algorithms (Encryption & MAC)	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC e.g., ECDSA)
Through 2010 (min. of 80 bits of strength)	2TDEA <sup>23</sup>  3TDEA  AES-128  AES-192  AES-256	Min.:  $L = 1024$ ; $N = 160$	Min.:  $k = 1024$	Min.:  $f = 160$
Through 2030 (min. of 112 bits of strength)	3TDEA  AES-128  AES-192  AES-256	Min.:  $L = 2048$ $N = 224$	Min.:  $k = 2048$	Min.:  $f = 224$
Beyond 2030 (min. of 128 bits of strength)	AES-128  AES-192  AES-256	Min.:  $L = 3072$ $N = 256$	Min.:  $k = 3072$	Min.:  $f = 256$

NIST SP 800-57,  
"Recommendation for Key Management, Part 1: General (Revised)", Marzo 2007

# Che parametri scegliere?

Bisogna tener conto dell'  
*expected security life*

Esempi:

- Fatto nel 2025
- Expected security life = 5 anni →
- Fatto nel 2025
- Expected security life = 6 anni →

Table 4: Recommended algorithms and minimum key sizes

Algorithm security lifetimes	Symmetric key algorithms (Encryption & MAC)	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC e.g., ECDSA)
Through 2010 (min. of 80 bits of strength)	2TDEA <sup>23</sup> 3TDEA AES-128 AES-192 AES-256	Min.: $L = 1024$ ; $N = 160$	Min.: $k = 1024$	Min.: $f = 160$
Through 2030 (min. of 112 bits of strength)	3TDEA AES-128 AES-192 AES-256	Min.: $L = 2048$ $N = 224$	Min.: $k = 2048$	Min.: $f = 224$
Beyond 2030 (min. of 128 bits of strength)	AES-128 AES-192 AES-256	Min.: $L = 3072$ $N = 256$	Min.: $k = 3072$	Min.: $f = 256$

NIST SP 800-57,

"Recommendation for Key Management, Part 1: General (Revised)", Marzo 2007

# Che parametri scegliere?

Security Strength	Symmetric key algorithms	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC (e.g., ECDSA)
≤ 80	2TDEA	$L = 1024$ $N = 160$	$k = 1024$	$f = 160-223$
112	3TDEA	$L = 2048$ $N = 224$	$k = 2048$	$f = 224-255$
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256-383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384-511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$



Non approvati per la protezione di informazioni del Governo Federale USA



Inclusi negli standard NIST



Non ancora inclusi negli standard NIST per motivi di interoperabilità e di efficienza

NIST SP 800-57 Part 1 Revision 4 (pp. 52-53)

"Recommendation for Key Management Part 1: General", Gennaio 2016

<http://dx.doi.org/10.6028/NIST.SP.800-57pt1r4>

# Equivalenza key size

Come si stabilisce l'equivalenza  
(per la sicurezza) tra le  
lunghezze delle chiavi  
pubbliche e simmetriche?



# Equivalenza key size

Come si stabilisce l'equivalenza  
(per la sicurezza) tra le  
lunghezze delle chiavi  
pubbliche e simmetriche?



Risolvere l'equazione:

$$2^k = \text{complessità GNFS } (2^N)$$



Sicurezza cifrario a  
blocchi con chiave di  $k$  bit



Sicurezza RSA per  $n$  di  
lunghezza  $N = \log_2 n$

# Equivalenza key size

Complessità GNFS:  $O(e^{(c+o(1))(\ln n)^{1/3}(\ln \ln n)^{2/3}})$

# Equivalenza key size

Complessità GNFS:  $O(e^{(c+o(1))(\ln n)^{1/3}(\ln \ln n)^{2/3}})$

quindi:  $Ae^{(c+o(1))(\ln n)^{1/3}(\ln \ln n)^{2/3}}$

da determinare

$$c = \left(\frac{64}{9}\right)^{1/3} = 1.922999$$

= 0

# Equivalenza key size

Complessità GNFS:  $O(e^{(c+o(1))(\ln n)^{1/3}(\ln \ln n)^{2/3}})$

quindi:  $Ae^{(c+o(1))(\ln n)^{1/3}(\ln \ln n)^{2/3}}$

da determinare

$$c = \left(\frac{64}{9}\right)^{1/3} = 1.922999$$

= 0

$$k = \log_2 Ae^{c(\ln n)^{1/3}(\ln \ln n)^{2/3}}$$

$$= \log_2 Ae^{c(\ln 2^N)^{1/3}(\ln \ln 2^N)^{2/3}}$$

sostituendo  $n=2^N$

$$= \log_2 A + (64/9)^{1/3} \log_2(e) (N \ln 2)^{1/3} (\ln(N \ln 2))^{2/3}$$

# Equivalenza key size

Complessità GNFS:  $O(e^{(c+o(1))(\ln n)^{1/3}(\ln \ln n)^{2/3}})$

quindi:  $Ae^{(c+o(1))(\ln n)^{1/3}(\ln \ln n)^{2/3}}$

da determinare

$$c = \left(\frac{64}{9}\right)^{1/3} = 1.922999 = 0$$

$$\begin{aligned} k &= \log_2 Ae^{c(\ln n)^{1/3}(\ln \ln n)^{2/3}} \\ &= \log_2 Ae^{c(\ln 2N)^{1/3}(\ln \ln 2N)^{2/3}} \\ &= \log_2 A + (64/9)^{1/3} \log_2(e) (N \ln 2)^{1/3} (\ln(N \ln 2))^{2/3} \end{aligned}$$

Come si determina la costante A?

# Equivalenza key size

$2^k = \text{complessità GNFS } (2^N)$

$$k = \log_2 A + (64/9)^{1/3} \log_2(e) (N \ln 2)^{1/3} (\ln(N \ln 2))^{2/3}$$

**Determinazione di A:** Per interi di 512 bit la complessità di GNFS è 4-6 bit meno di DES (56 bit). Assumiamo 50 bit (approccio più conservativo).

$$k = (64/9)^{1/3} \log_2(e) (N \ln 2)^{1/3} (\ln(N \ln 2))^{2/3} - 14$$

# Equivalenza key size

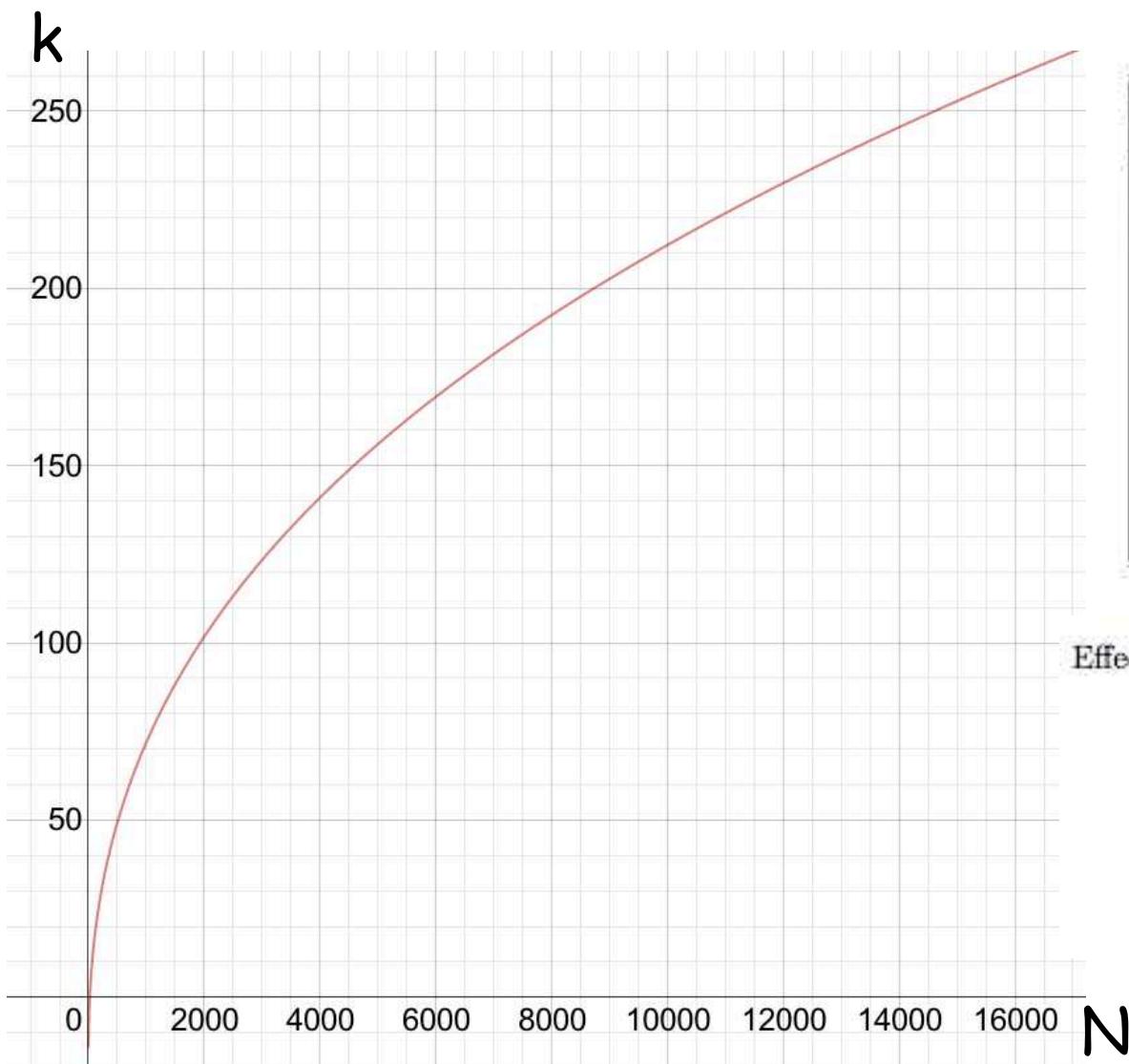


Table 7.2: Key-size Equivalence.

Security (bits)	RSA	DLOG		EC
		field size	subfield	
48	480	480	96	96
56	640	640	112	112
64	816	816	128	128
80	1248	1248	160	160
112	2432	2432	224	224
128	3248	3248	256	256
160	5312	5312	320	320
192	7936	7936	384	384
256	15424	15424	512	512

Effective Key-size of Commonly used RSA/DLOG Keys.

RSA/DLOG Key	Security (bits)
512	50
768	62
1024	73
1536	89
2048	103

# Sicurezza cifratura RSA



Conoscendo la chiave pubblica  $(n,e)$  e il messaggio cifrato  $C \leftarrow M^e \text{ mod } n$



vuole calcolare il messaggio  $M$

# Sicurezza cifratura RSA

Se  potesse fattorizzare n  
potrebbe anche computare M

1. Fattorizza n
2. Computa  $\varphi(n) = (p-1)(q-1)$
3. Computa  $d \leftarrow e^{-1} \pmod{(p-1)(q-1)}$
4. Ricava M decifrando C

# Sicurezza cifratura RSA

Se  potesse computare  $M$  ...

Importante problema aperto:

non si sa se questo sia **computazionalmente**  
**equivalente** a fattorizzare!

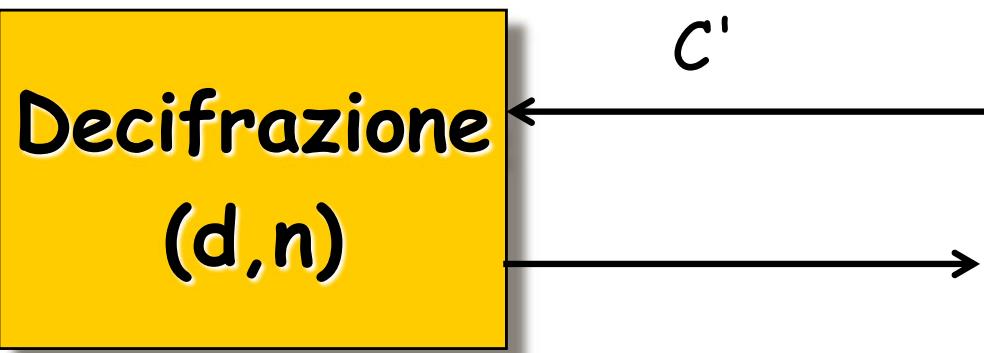
# Altri attacchi ad RSA

- Attacchi non basati sul problema della fattorizzazione
  - Chosen ciphertext attack
  - Common modulus attack
  - Low exponent attack
- Attacchi ad implementazioni



# Chosen ciphertext attack

Obiettivo: decifrare  $C$  ( $= M^e \text{ mod } n$ )



Calcolo  $C'$  da  $C$



# Chosen ciphertext attack

Obiettivo: decifrare  $C$  ( $= M^e \text{ mod } n$ )

Decifrazione  
( $d, n$ )

$C'$

$M'$

Calcolo  $C'$  da  $C$



Ora posso calcolare  $M$

Calcolo  $M$  da  $M'$

# Chosen ciphertext attack

$$\left. \begin{array}{l} C_1 = M_1^e \bmod n \\ C_2 = M_2^e \bmod n \end{array} \right\}$$

$$(M_1 \cdot M_2)^e = M_1^e \cdot M_2^e = C_1 \cdot C_2 \bmod n$$

Proprietà di omomorfismo

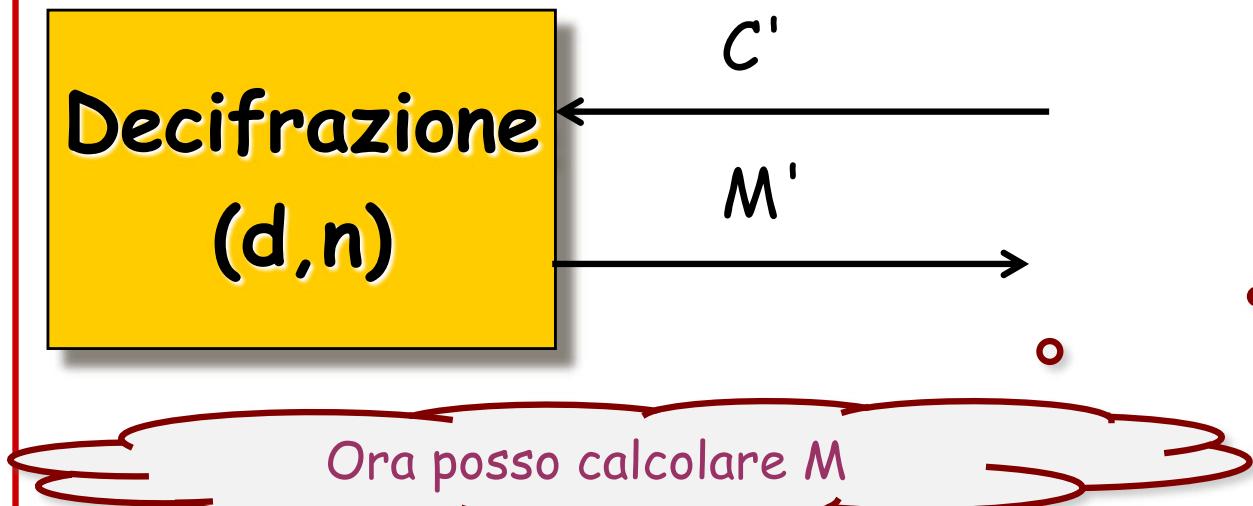
# Chosen ciphertext attack

$$\begin{aligned} C_1 &= M_1^e \bmod n \\ C_2 &= M_2^e \bmod n \end{aligned}$$

$$(M_1 \cdot M_2)^e = M_1^e \cdot M_2^e = C_1 \cdot C_2 \bmod n$$

Proprietà di omomorfismo

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



Calcolo  $C'$  da  $C$



Calcolo  $M$  da  $M'$

# Chosen ciphertext attack

$$\left. \begin{array}{l} C_1 = M_1^e \bmod n \\ C_2 = M_2^e \bmod n \end{array} \right\}$$

$$(M_1 \cdot M_2)^e = M_1^e \cdot M_2^e = C_1 \cdot C_2 \bmod n$$

Proprietà di omomorfismo

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

Decifrazione  
( $d, n$ )

$$C' \leftarrow C \cdot x^e \bmod n$$

$$M' \leftarrow (C')^d \bmod n$$

Scelgo  $x$  a caso



Ora posso calcolare  $M$

Calcolo  $M$  da  $M'$

# Chosen ciphertext attack

$$\left. \begin{array}{l} C_1 = M_1^e \bmod n \\ C_2 = M_2^e \bmod n \end{array} \right\}$$

$$(M_1 \cdot M_2)^e = M_1^e \cdot M_2^e = C_1 \cdot C_2 \bmod n$$

Proprietà di omomorfismo

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

Decifrazione  
( $d, n$ )

$$C' \leftarrow C \cdot x^e \bmod n$$

$$M' \leftarrow (C')^d \bmod n$$

Scelgo  $x$  a caso



$$M' = (C')^d = (C \cdot x^e)^d = C^d \cdot x^d = M \cdot x \bmod n$$

Calcolo  $M$  da  $M'$

# Chosen ciphertext attack

$$\left. \begin{array}{l} C_1 = M_1^e \bmod n \\ C_2 = M_2^e \bmod n \end{array} \right\}$$

$$(M_1 \cdot M_2)^e = M_1^e \cdot M_2^e = C_1 \cdot C_2 \bmod n$$

Proprietà di omomorfismo

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

Decifrazione  
( $d, n$ )

$$C' \leftarrow C \cdot x^e \bmod n$$

$$M' \leftarrow (C')^d \bmod n$$

Scelgo  $x$  a caso



$$M' = (C')^d = (C \cdot x^e)^d = C^d \cdot x^d = M \cdot x \bmod n$$

$$M \leftarrow M' \cdot x^{-1} \bmod n$$

# Common Modulus Attack

- Stesso modulo  $n$  per diverse chiavi pubbliche
  - Chiave Alice:  $(n, e_1)$ , chiave Bob:  $(n, e_2)$
  - Supponiamo che  $\gcd(e_1, e_2) = 1$
- Stesso messaggio  $M$  inviato ai vari utenti
  - Cifratura per Alice:  $C_1 = M^{e_1} \text{ mod } n$ ,
  - Cifratura per Bob:  $C_2 = M^{e_2} \text{ mod } n$
- E' semplice risalire ad  $M$ 
  - Usa Euclide-esteso per calcolare  $x, y$  tali che  $1 = e_1 \cdot x + e_2 \cdot y$
  - $C_1^x \cdot C_2^y \text{ mod } n = (M^{e_1})^x \cdot (M^{e_2})^y = M^{e_1x + e_2y} = M$

# Low Exponent Attack

- Stesso  $e$  per diverse chiavi pubbliche
  - Chiave Alice:  $(n_1, 3)$ , chiave Bob:  $(n_2, 3)$ , chiave Eva:  $(n_3, 3)$
  - Supponiamo che  $\gcd(n_i, n_j) = 1, i \neq j$
- Stesso messaggio  $M$  inviato ai vari utenti
  - Cifratura per Alice:  $C_1 = M^3 \pmod{n_1}$
  - Cifratura per Bob:  $C_2 = M^3 \pmod{n_2}$
  - Cifratura per Ciro:  $C_3 = M^3 \pmod{n_3}$
- E' semplice risalire ad  $M$ 
  - Usa Teorema cinese del resto per calcolare la soluzione di
$$\begin{cases} x \equiv C_1 \pmod{n_1} \\ x \equiv C_2 \pmod{n_2} \\ x \equiv C_3 \pmod{n_3} \end{cases}$$
$$x = M^3 \pmod{n_1 \cdot n_2 \cdot n_3}$$

poi calcola  $M = x^{1/3}$

# RSA: Attacchi ad implementazioni

- Timing Attack [Kocher, '97]
  - Ricava i bit di  $d$  uno alla volta, analizzando il tempo richiesto per l'esponenziazione modulare (decifratura)
- Power Attack [Kocher, '99]
  - Ricava  $d$  analizzando la potenza consumata da una smartcard durante la decifratura

# RSA: Attacchi ad implementazioni

- Timing Attack [Kocher, '97]
  - Ricava i bit di  $d$  uno alla volta, analizzando il tempo richiesto per l'esponenziazione modulare (decifratura)
- Power Attack [Kocher, '99]
  - Ricava  $d$  analizzando la potenza consumata da una smartcard durante la decifratura
- Contromisure
  - Ritardo costante (tutte le esponenziazioni richiedono lo stesso tempo)
  - Ritardo casuale (introduce "rumore" per confondere l'avversario)
  - Blinding (moltiplica il cfrato per un numero casuale prima di decifrare)

# RSA: Attacchi ad implementazioni

- Timing Attack [Kocher, '97]
  - Ricava i bit di  $d$  uno alla volta, analizzando il tempo richiesto per l'esponenziazione modulare (decifratura)
- Power Attack [Kocher, '99]
  - Ricava  $d$  analizzando la potenza consumata da una smartcard durante la decifratura
- Contromisure
  - Ritardo costante (tutte le esponenziazioni richiedono lo stesso tempo)
  - Ritardo casuale (introduce "rumore" per confondere l'avversario)
  - Blinding (moltiplica il cifrato  $C^d \text{ mod } n$  per una chiave di decifrare)

Scegli a caso  $x$

$$M = (Cx^e)^d \cdot x^{-1} \text{ mod } n$$

# Sommario

- RSA
- Descrizione
- Generazione dei parametri
- Sicurezza
  - Attacchi
  - Fattorizzazione
  - Lunghezza della chiave
- Uso in pratica

# Cifrari simmetrici e asimmetrici

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  $\geq 100$  volte più veloce di RSA, in hardware tra 1.000 e 10.000 volte)
- Sufficiente in diverse situazioni  
(ad esempio, applicazioni per singolo utente)

# Cifrari simmetrici e asimmetrici

Vantaggi della crittografia a chiave pubblica

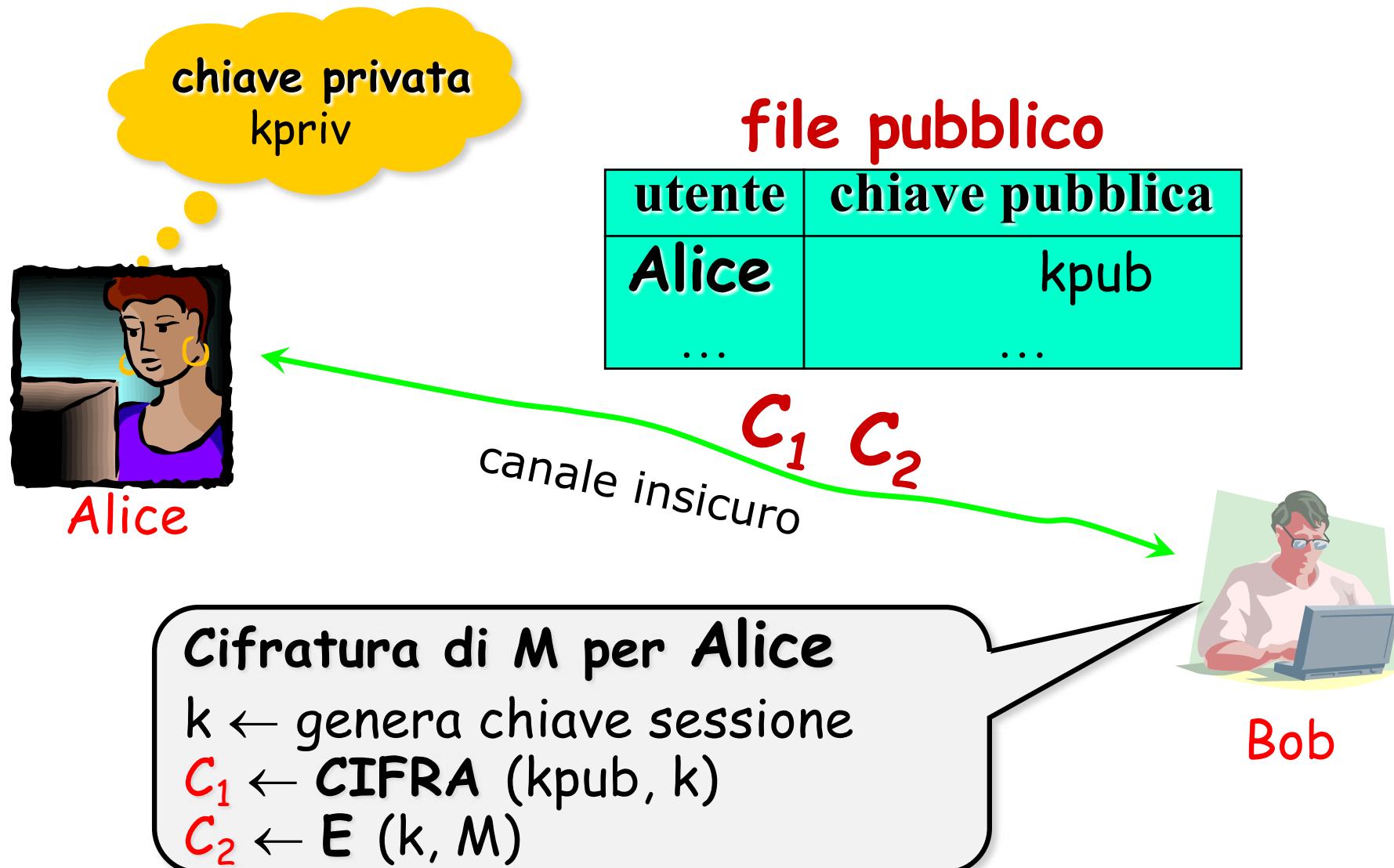
- Chiavi private mai trasmesse
- Possibile la firma digitale

Vantaggi della crittografia a chiave privata

- Molto più veloce (ad es., DES è  $\geq 100$  volte più veloce di RSA, in hardware tra 1.000 e 10.000 volte)
- Sufficiente in diverse situazioni  
(ad esempio, applicazioni per singolo utente)

Ci sono anche alcuni problemi di sicurezza ...

# Cifrari ibridi



# Cifrari ibridi

chiave privata  
kpriv

file pubblico

utente	chiave pubblica
Alice	kpub
...	...



Decifratura di  $C_1, C_2$   
 $k \leftarrow \text{DECIFRA}(\text{kpriv}, C_1)$   
 $M \leftarrow D(k, C_2)$

$C_1, C_2$

# Terminologia

Documenti NIST:  
asymmetric-key-based key transport scheme

- **Key transport:** A key establishment procedure whereby one party (the sender) selects a value for the secret keying material and then securely distributes that value to another party (the receiver). Contrast with key agreement.

# Uso di RSA

- Textbook RSA
- Poco sicuro

# Un semplice attacco su textbook RSA

- Cifratura di una chiave di sessione di 64 bit       $C = K^e \text{ mod } n$
- Supponiamo  $K = K_1 \cdot K_2$  con  $K_1, K_2 < 2^{34}$       (probabilità  $\approx 20\%$ )

# Un semplice attacco su textbook RSA

- Cifratura di una chiave di sessione di 64 bit       $C = K^e \text{ mod } n$
  - Supponiamo  $K = K_1 \cdot K_2$  con  $K_1, K_2 < 2^{34}$       (probabilità  $\approx 20\%$ )
  - Allora     $C/K_1^e = K_2^e \text{ mod } n$

# Un semplice attacco su textbook RSA

- Cifratura di una chiave di sessione di 64 bit       $C = K^e \text{ mod } n$
- Supponiamo  $K = K_1 \cdot K_2$  con  $K_1, K_2 < 2^{34}$       (probabilità  $\approx 20\%$ )
- Allora     $C/K_1^e = K_2^e \text{ mod } n$

Decifra ( $C = K^e \text{ mod } n$ )

Costruiamo tabella:

$1^e$	$C/1^e$
$2^e$	$C/2^e$
$3^e$	$C/3^e$
...	...
$2^{34}e$	$C/2^{34}e$

For  $K_2 = 0, \dots, 2^{34}$

if  $K_2^e$  è nella tabella, "chiave trovata"

# Un semplice attacco su textbook RSA

- Cifratura di una chiave di sessione di 64 bit       $C = K^e \text{ mod } n$
- Supponiamo  $K = K_1 \cdot K_2$  con  $K_1, K_2 < 2^{34}$       (probabilità  $\approx 20\%$ )
- Allora     $C/K_1^e = K_2^e \text{ mod } n$

Decifra ( $C = K^e \text{ mod } n$ )

Costruiamo tabella:

$1^e$	$C/1^e$
$2^e$	$C/2^e$
$3^e$	$C/3^e$
...	...
$2^{34}e$	$C/2^{34}e$

tempo  $2^{34}$

For  $K_2 = 0, \dots, 2^{34}$

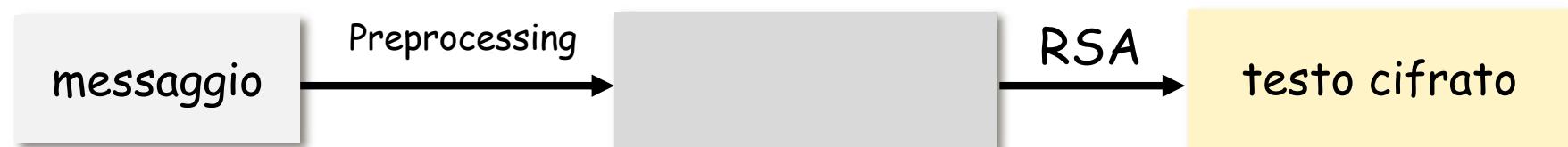
if  $K_2^e$  è nella tabella, "chiave trovata"

tempo  $2^{34} \cdot 34$

Tempo totale  $\approx 2^{40} \ll 2^{64}$

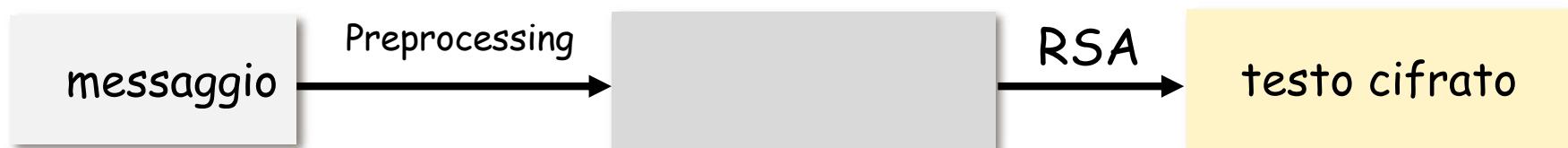
# Uso di RSA

- Textbook RSA
- Poco sicuro
- RSA in pratica:



# Uso di RSA

- Textbook RSA
- Poco sicuro
- RSA in pratica:

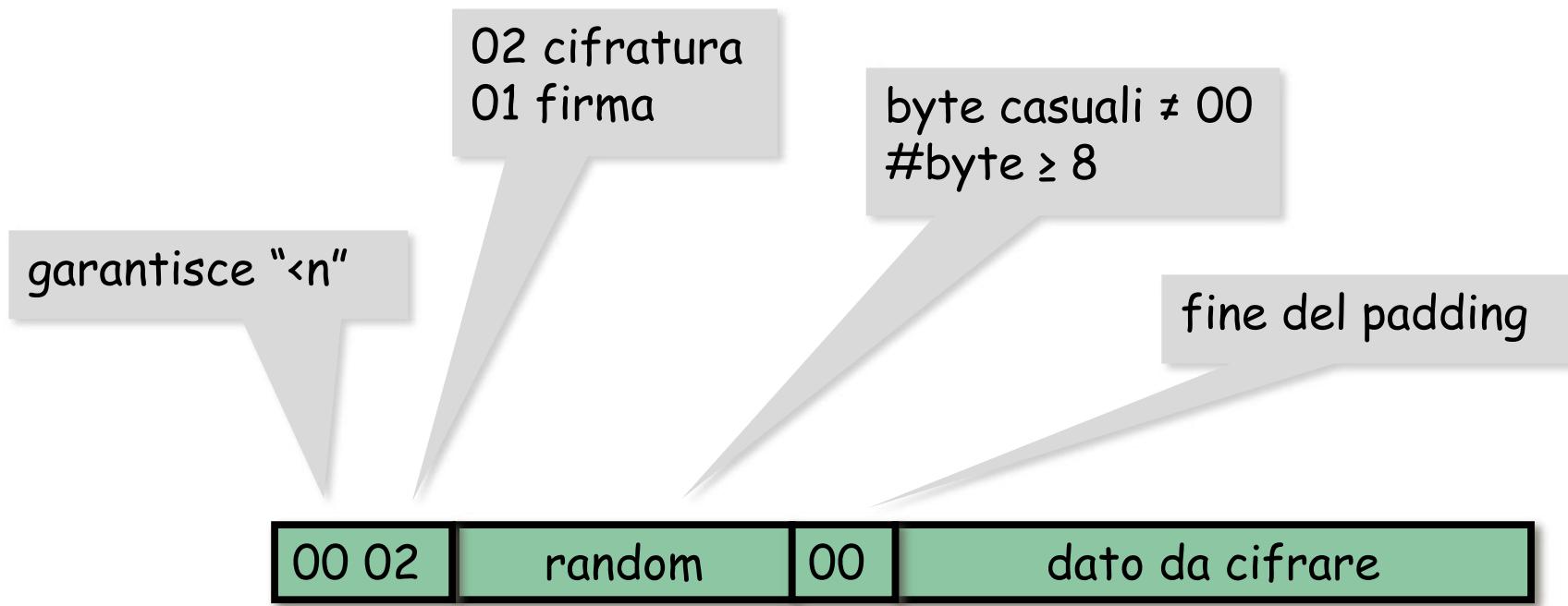


Come fare il preprocessing?

# PKCS #1

- Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications, Version 2.1
- Ver. 1.4, giugno 1991, ..., Ver. 2.1, giugno 2002 (= RFC 3447 nel feb 2003)
- Due schemi per la cifratura
  - RSAES-PKCS1-v1\_5  
Problemi di sicurezza
  - RSAES-OAEP. (Optimal Asymmetric Encryption Padding)  
Rende casuale il messaggio da cifrare  
M. Bellare and P. Rogaway.  
Optimal Asymmetric Encryption - How to Encrypt with RSA,  
Eurocrypt 1994
- Provable security in the random oracle model

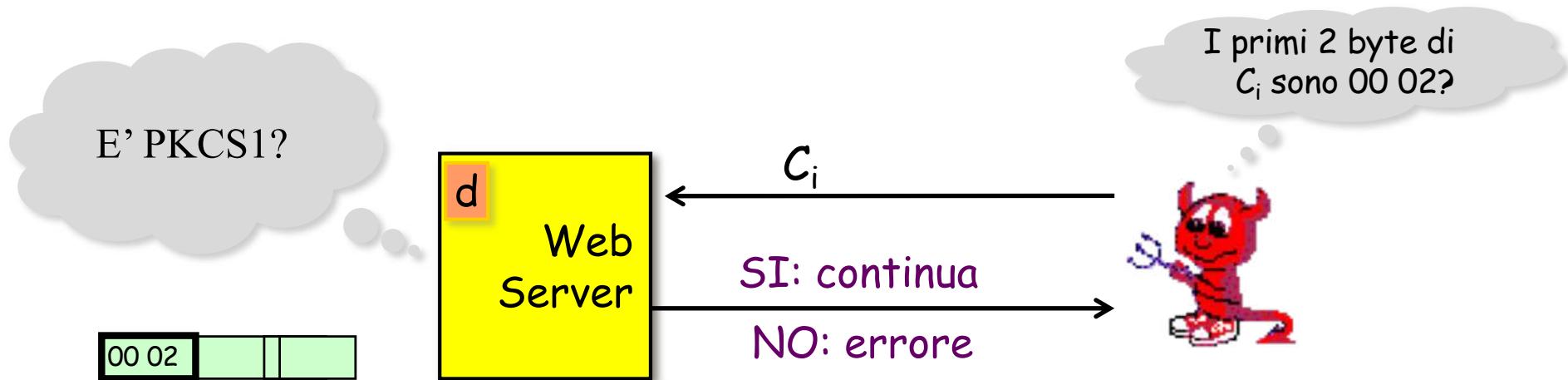
# RSAES-PKCS1-v1\_5



Cifratura(dato da cifrare) = ( )<sup>e</sup> mod n

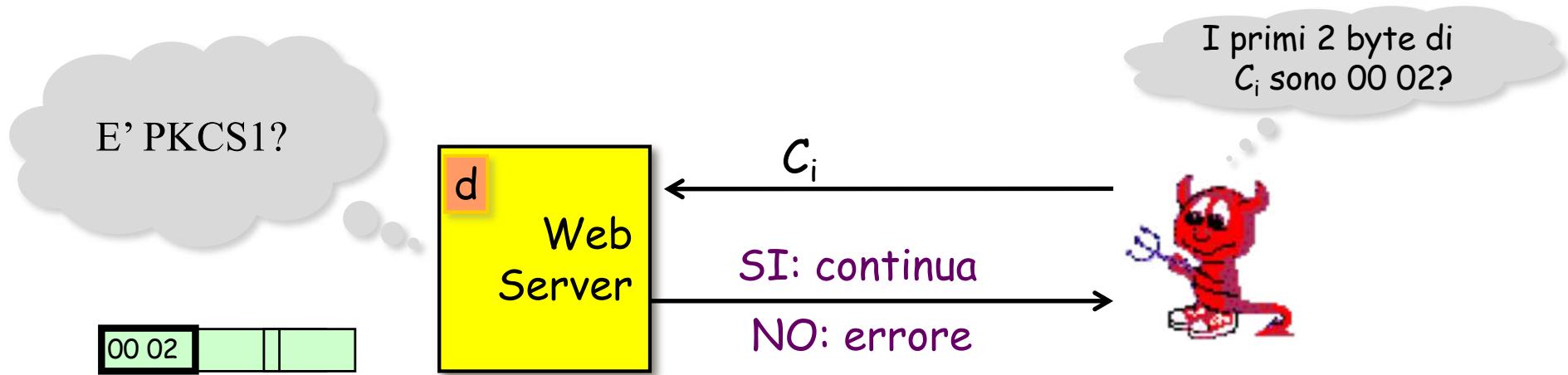
# Un attacco a PKCS1 usato in SSL

Bleichenbacher [1998], chosen ciphertext attack



# Un attacco a PKCS1 usato in SSL

Bleichenbacher [1998], chosen ciphertext attack



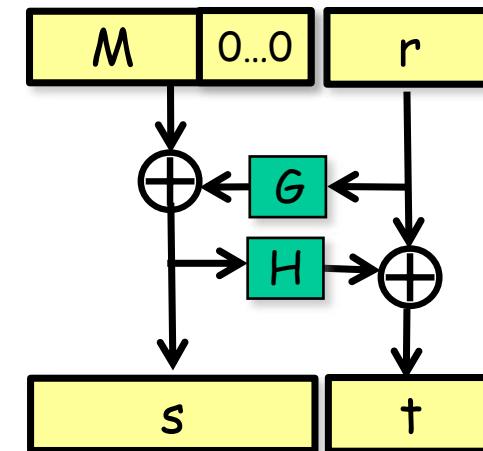
Per decifrare un testo cifrato  $C = (\text{PKCS1}(M))^e \bmod n$ :

- Scegliere a caso  $r_i \in \mathbb{Z}_n$
- $C_i = r_i^e \cdot C = (r_i \cdot \text{PKCS1}(M))^e \bmod n$
- Invia  $C_i$  al server web ed annota la risposta
  - Se è accettato allora i primi 2 byte di  $r_i \cdot \text{PKCS1}(M) \bmod n$  sono = 00 02  
altrimenti sono  $\neq 00 02$

# Optimal Asymmetric Encryption Padding

OAEP → Cifratura

- $s = (M||0...0) \oplus G(r)$
  - $t = r \oplus H(s)$
  - $C = \text{Enc}(s||t)$
- } padding      } cifratura



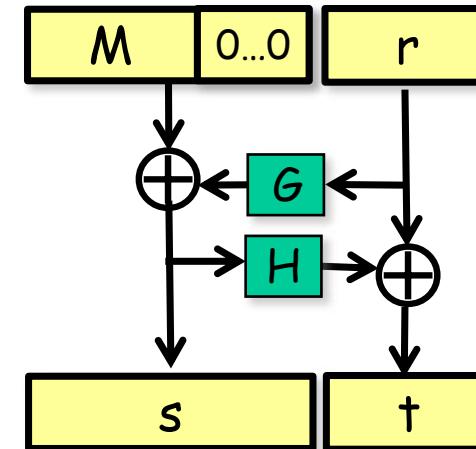
funzioni hash

Struttura simile ad un cifrario Feistel

# Optimal Asymmetric Encryption Padding

OAEP  $\rightarrow$  Cifratura

- $s = (M||0...0) \oplus G(r)$
  - $t = r \oplus H(s)$
  - $C = \text{Enc}(s||t)$
- } padding      } cifratura



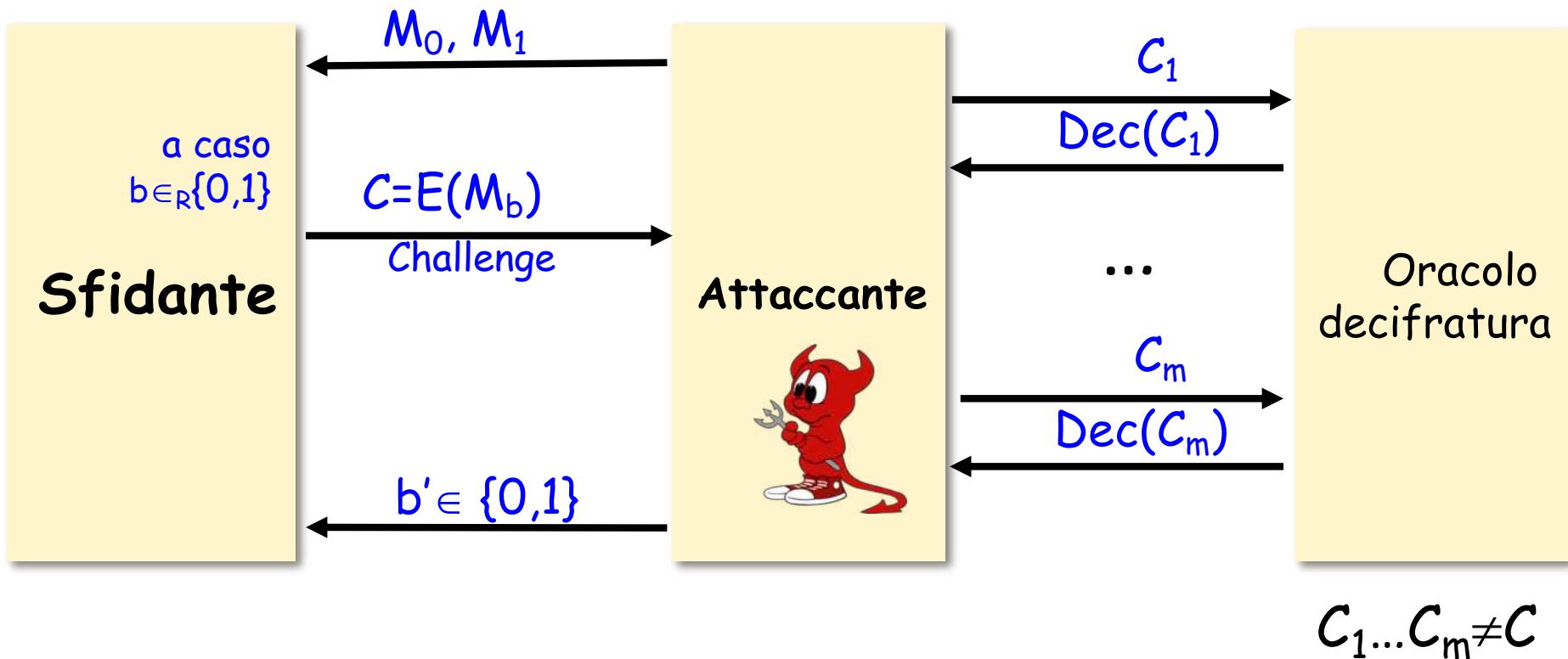
$$C = ( \ )^e \bmod n$$

$G$   $H$  modellate  
come random oracle

cifratura RSA con OAEP  
sicura rispetto ad  
attacchi CCA

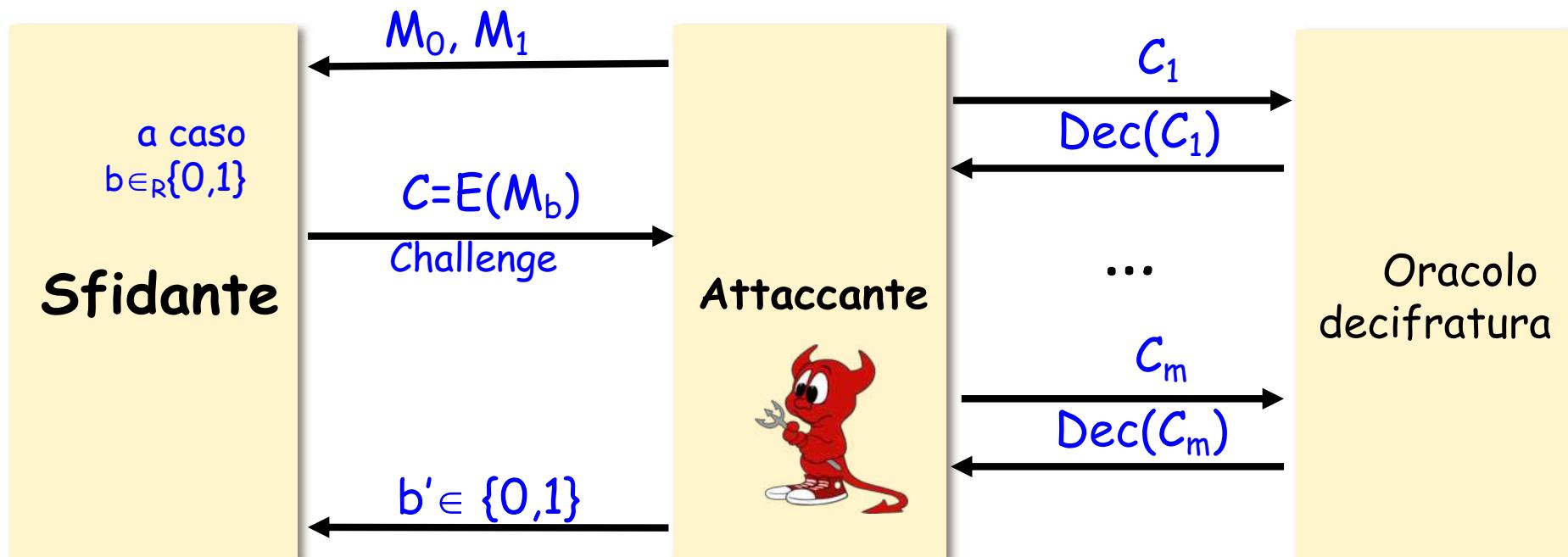
# Chosen Ciphertext Attack (CCA)

Nessun attaccante efficiente può vincere questo gioco con probabilità non-insignificante



# Chosen Ciphertext Attack (CCA)

Nessun attaccante efficiente può vincere questo gioco con probabilità non-insignificante



Attaccante vince se  $b=b'$

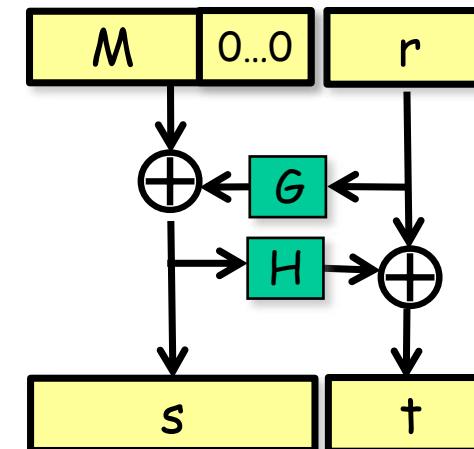


$$C_1 \dots C_m \neq C$$

# Optimal Asymmetric Encryption Padding

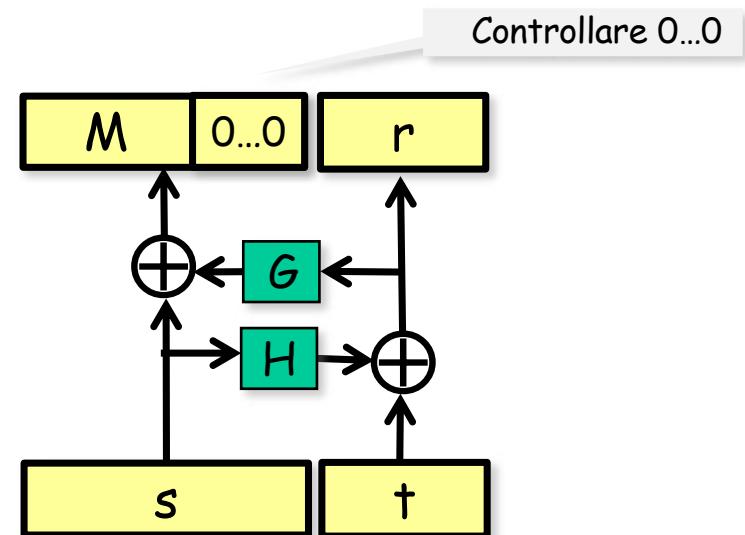
OAEP → Cifratura

- $s = (M||0...0) \oplus G(r)$
  - $t = r \oplus H(s)$
  - $C = \text{Enc}(s||t)$
- } padding                                   } cifratura

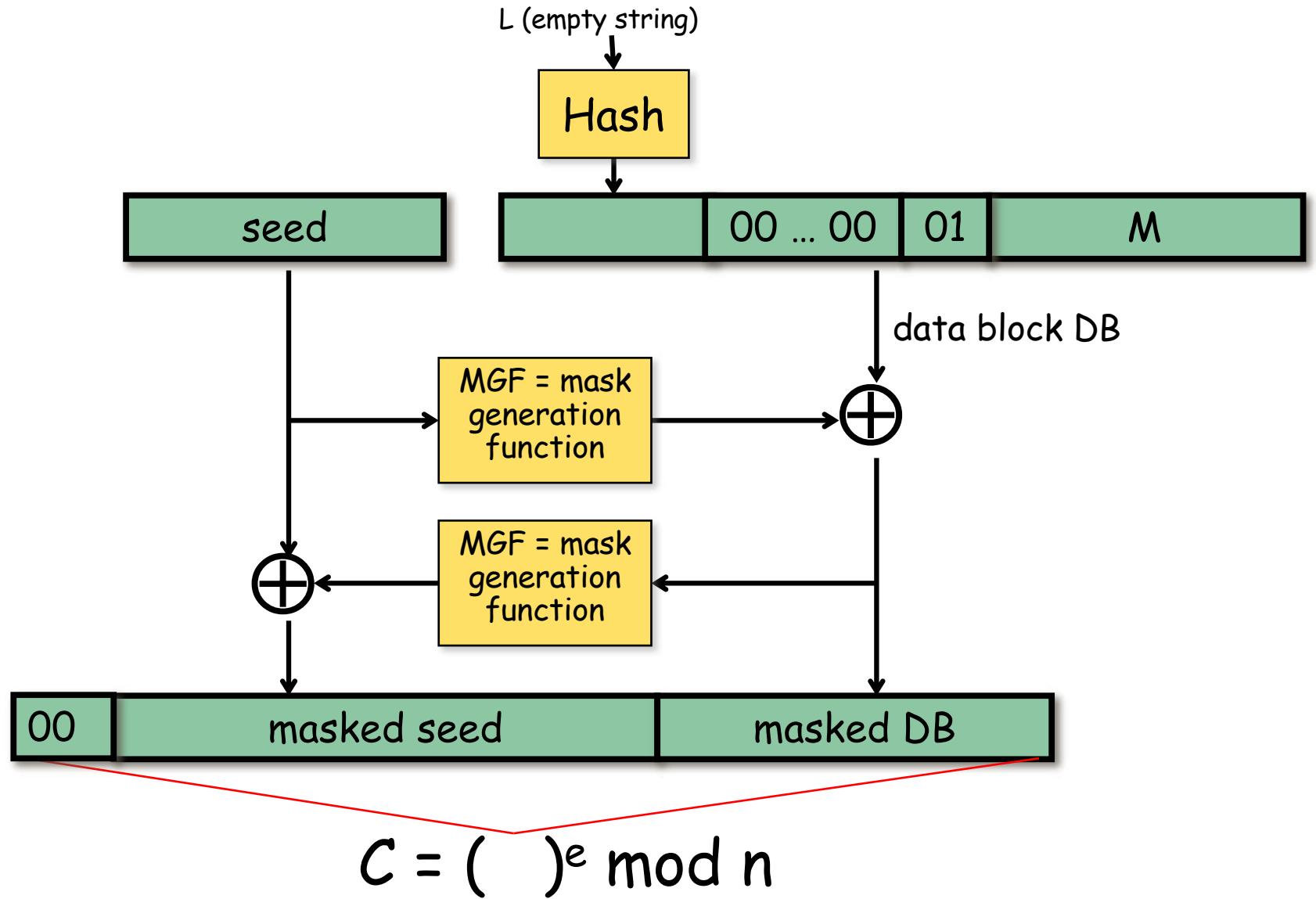


Decifratura → OAEP

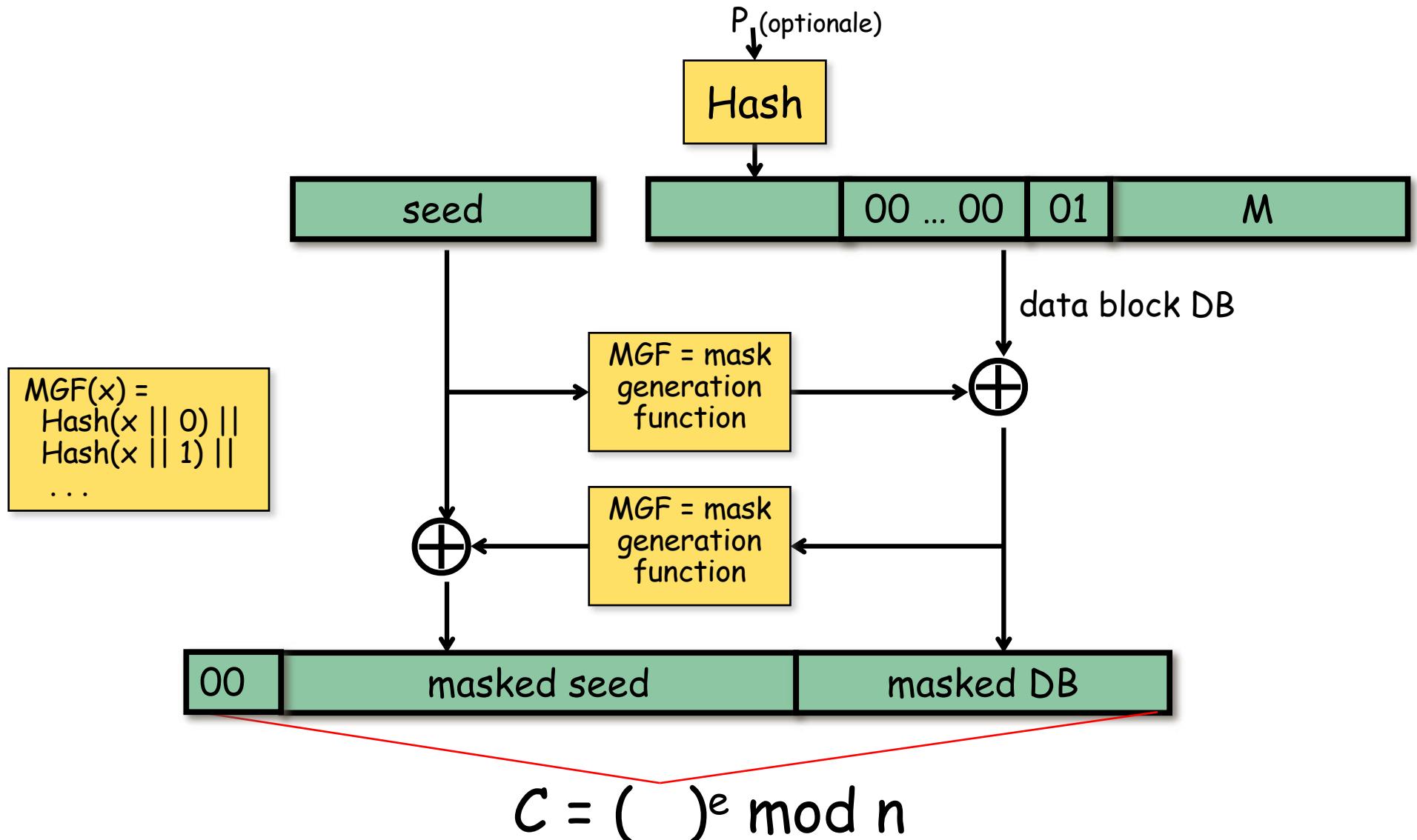
- $(s,t) = \text{Dec}(C)$
  - $r = t \oplus H(s)$
  - $M||0...0 = s \oplus G(r)$
- } decifratura                                   } padding



# RSA Encryption with RSA-OAEP Padding



# RSA Encryption with RSA-OAEP Padding



# Cifrari asimmetrici

- RSA [1977] (fattorizzazione)
- Merkle-Hellman [1978] (zaino 0-1) 
  - 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]
- Uso di curve iperellittiche [1989]
- Uso di automi cellulari [1985]
- ...

# Post-Quantum Cryptography

- Nei prossimi anni, possibile costruzione di Computer Quantistici che implementano l'algoritmo di Shor
  - Computazione efficiente di Fattorizzazione e Logaritmi Discreti
- Processo scelta algoritmi del NIST
  - Iniziato nel 2016
  - Standard disponibile nel 2022/2024
  - <https://csrc.nist.gov/projects/post-quantum-cryptography>
  - Proposte:
    - lattice-based cryptosystems
    - code-based cryptosystems
    - multivariate cryptosystems
    - hash-based signatures

# Crittografia a chiave pubblica

Nel 1997 è stato rivelato che è stata sviluppata da

- James H. Ellis, Clifford Cocks e Malcolm Williamson



- Government Communications Headquarters

(GCHQ), UK nel 1973

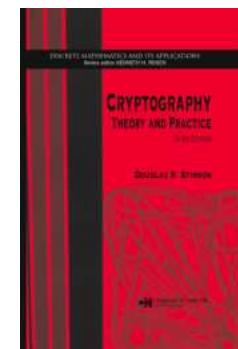


- Scoperta indipendente di accordo su chiavi Diffie-Hellman e caso speciale di RSA



# Bibliografia

- **Cryptography and Network Security**  
by W. Stallings, 2010
  - cap. 9 (Public-Key Cryptography and RSA)
- **Cryptography: Theory and Practice (I ed.)**  
by D.R. Stinson (1995)
  - cap 5 (The RSA System and Factoring)



# Domande?

