

Schemi basati sulla assunzione RSA

Plain RSA (1978, Rivest, Shamir, Adleman)

ALGORITHM 11.25

RSA key generation GenRSA

Input: Security parameter 1^n

Output: N, e, d as described in the text

$(N, p, q) \leftarrow \text{GenModulus}(1^n)$

$\phi(N) := (p - 1)(q - 1)$

choose $e > 1$ such that $\gcd(e, \phi(N)) = 1$

compute $d := [e^{-1} \bmod \phi(N)]$

return N, e, d

$$\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$$

CONSTRUCTION 11.26

Let GenRSA be as in the text. Define a public-key encryption scheme as follows:

- **Gen:** on input 1^n run GenRSA(1^n) to obtain N, e , and d . The public key is $\langle N, e \rangle$ and the private key is $\langle N, d \rangle$.
- **Enc:** on input a public key $pk = \langle N, e \rangle$ and a message $m \in \mathbb{Z}_N^*$, compute the ciphertext

$$c := [m^e \bmod N].$$

- **Dec:** on input a private key $sk = \langle N, d \rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute the message

$$m := [c^d \bmod N].$$

Perchè funziona la decifratura?

Ricordando il teorema di Eulero

$$m^{ed} = m^{1+h\varphi(\mathbf{N})} = m(m^{\varphi(\mathbf{N})})^h \equiv m(1)^h \equiv m \pmod{\mathbf{N}}.$$

In alternativa, in generale, ricordando i risultati sui gruppi finiti delle lezioni passate ...

Esempio gioca Alice

$$(N, e, d) \leftarrow \text{fun RSA}()$$

$$(391, 3, 235)$$



$$17 \cdot 23 \rightarrow \varphi(391) = 16 \cdot 22 = 352 \Rightarrow 3 \cdot 235 \equiv 1 \pmod{352}$$

$$p \quad q$$

$$pk = \langle 391, 3 \rangle$$

$$sk = \langle 391, 235 \rangle$$

Per cifrare $m = 158 \in \mathbb{Z}_{391}^*$ basta calcolare:

$$c = [158^3 \pmod{391}] = 295$$

Per decifrare

$$[295^{235} \pmod{391}] = 158$$

Circa le sicurezze ..

Fattorizzazione difficile \Rightarrow è computazionalmente inammissibile calcolare la chiave privata dalla pubblica

Problema RSA difficile \Rightarrow se il messaggio m è scelto unif. a caso in \mathbb{Z}_N^* , da c, N e
è computazionalmente inammissibile
calcolare m .

Tuttavia sono garanzie deboli rispetto alle nozioni di
sicurezza che vogliamo ottenere.

Può capire ...

- se il msg m non è scelto uniformemente in \mathbb{Z}_N^* ?
- se AdE è interessato ad informazioni parziali?

Inoltre, RSA-plain è deterministico

- stesso messaggio / stesso cifrato

Sono noti attacchi generici e specifici:

- algoritmi che passano per la fattorizzazione di intero time
- algoritmi che sfruttano "e" piccolo
- algoritmi che sfruttano informazioni parziali su m
- algoritmi che sfruttano messaggi relativi o cifratelli multipli

RSA randomization

CONSTRUCTION 11.30

Let GenRSA be as before, and let ℓ be a function with $\ell(n) \leq 2n - 4$ for all n . Define a public-key encryption scheme as follows:

- **Gen:** on input 1^n , run GenRSA(1^n) to obtain (N, e, d) . Output the public key $pk = \langle N, e \rangle$, and the private key $sk = \langle N, d \rangle$.
- **Enc:** on input a public key $pk = \langle N, e \rangle$ and a message $m \in \{0, 1\}^{\|N\| - \ell(n) - 2}$, choose a uniform string $r \in \{0, 1\}^{\ell(n)}$ and interpret $\hat{m} := r \| m$ as an element of \mathbb{Z}_N^* . Output the ciphertext

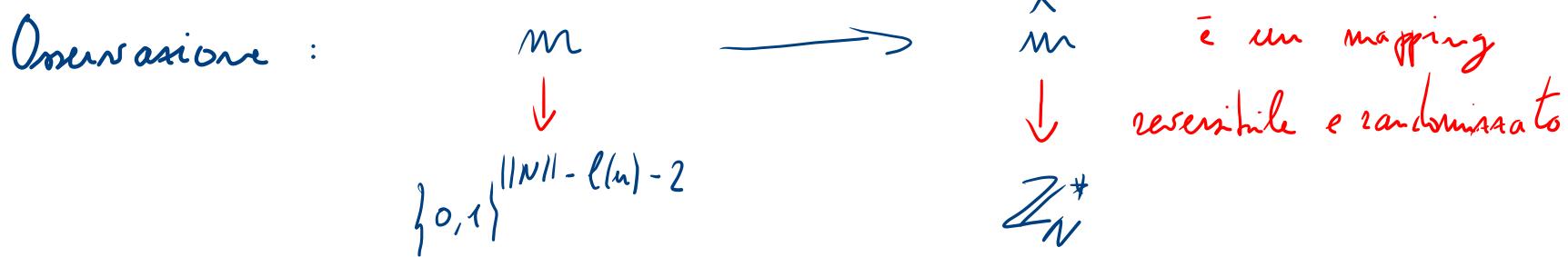
$$c := [\hat{m}^e \bmod N].$$

- **Dec:** on input a private key $sk = \langle N, d \rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute

$$\hat{m} := [c^d \bmod N],$$

and output the $\|N\| - \ell(n) - 2$ least-significant bits of \hat{m} .

The padded RSA encryption scheme.



Inoltre, permette di vedere i messaggi come
stringhe binarie di lunghezza al più $||N|| - l(n) - 2$

La sicurezza di Padded RSA dipende da l .

Se l è troppo piccolo, possono essere applicati tutti i possibili valori di s ed applicato l'attacco di sputta la conoscenza parziale di \hat{m} per decifrare c .

D'altra parte, se il PAD è il più grande possibile, i.e., m è un singolo bit, allora Padded RSA è CPA-sicuro.

Per tutti i casi "intermedi", la situazione non è chiara.

- non ci sono prove di attacchi PPT^{no}
esistere x vale l'assunzione RSA,
momento non x ne conoscere.

RSA PKCS #1 v1.5

Standard creato nel 1993 dagli RSA lab

Utilizza una variante del padding descritto in precedenza

Precisamente, data $pk = \langle N, e \rangle$, definiamo

- K , lunghezza di N in byte, vale a dire

$$2^{8(K-1)} \leq N < 2^{8K}$$

- D , lunghezza del messaggio m in byte. Deve essere

$$1 \leq D \leq K - 11$$

La cifratura di un messaggio m allora dà:

$$C = \left(0x00 \parallel 0x02 \parallel z \parallel 0x00 \parallel m\right)^e \bmod N$$

3 byte ← | | | ↗ D byte
 not. ESADICIMALE ↗ stringa casuale di K-D-3 byte

Poiché $D \leq K - 11 \Rightarrow z$ è di ALMENO 8 byte

Purtroppo, non è CPA-sicuro !

Un Adv può calcolare la parte iniziale d un msg m di cui si sa che ha molti Ø alla fine

Per capire, sia $m = \underbrace{b \parallel 0 \dots 0}_L$, $b \in \{0, 1\}$
 NON NOTO

di lunghezza minima, i.e., $L = 8 \cdot (k-1) - 1$. Risulta

$$c = (0x00 \parallel 0x02 \parallel 2 \parallel 0x00 \parallel b \parallel 0 \dots 0)^e \bmod N$$

Allora puoi calcolare $c' = c / \binom{2^L}{2}^e \bmod N$. Risulta:

$$c' = \left(\frac{0x00 \parallel 0x02 \parallel 2 \parallel 0x00 \parallel b \parallel 0 \dots 0}{10 \dots 0} \right)^e = (0x00 \parallel 0x02 \parallel 2 \parallel 0x00 \parallel b)^e \bmod N$$

Senza contare gli zeri iniziali, $0x02 \parallel 2 \parallel 0x00 \parallel b$ è lungo 75 bit.

\downarrow \downarrow \downarrow \downarrow
 $z + 4$ t 8 $+ 1$ $\overbrace{}^1$

Pertanto, Adv applicando

- gli attacchi per messaggi brevi, oppure
- gli attacchi basati su conoscenza parziale di m
nese a calcolare b

Occorre, quindi, avere valori di $2 \frac{1}{2}$ "più grandi".

Se $2 \frac{1}{2}$ circa la metà del messaggio è congettura
che PKCS #1 v1.5 risulti CPA-sicuro. Tuttavia, è noto
un attacco serio di tipo CCA che ha indotto ad
introdurre nuovi standard.

Uno schema di cifratura CPA-sicuro basato sull'assunzione RSA

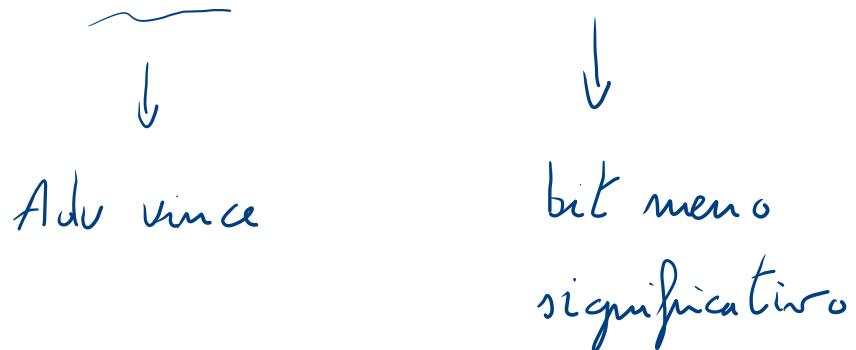
Procediamo in due passi:

- descriviamo un predicato Hard-core specifico per la permutazione RSA
- mostriamo come usare il predicato per cifrare un messaggio di un singolo bit.

Dato un algoritmo GenRSA ed un Aolv,
definiamo l'esperimento $\text{RSA}_{\text{lab}}^{(n)}$
 A, GenRSA

The RSA hard-core predicate experiment $\text{RSA-lsb}_{\mathcal{A}, \text{GenRSA}}(1^n)$:

1. Run $\text{GenRSA}(1^n)$ to obtain (N, e, d) .
2. choose a uniform $x \in \mathbb{Z}_N^*$ and compute $y := [x^e \bmod N]$.
3. \mathcal{A} is given N, e, y , and outputs a bit b .
4. The output of the experiment is 1 if and only if $\text{lsb}(x) = b$.



Nota: $\text{lsb}(x)$ è un bit uniforme quando
 $x \in \mathbb{Z}_N^*$ viene scelto uniformemente a caso.
A può indovinare $\text{lsb}(x)$ con prob. $1/2$.

Teorema RSA difficile rel. a GenRSA \Rightarrow $\forall A$ ppt

$\exists \text{negl}(n)$ tale che

$$P_2 \left[\underset{\lambda, \text{GenRSA}}{\text{RSA_lbb}}(n) = 1 \right] \leq \frac{1}{2} + \text{negl}(n)$$

In altre parole, $\text{lbb}(\cdot)$ è un predicato hardcore per la permutazione RSA.

Intuizione sulla prova del teorema.

Se esistesse un alg. efficiente di calcolo SEMPRE

$\text{lbb}(x)$ da N , e $x^e \bmod N$, allora potrebbe essere usato per calcolare efficientemente x da N , e $x^e \bmod N$
(... invertire RSA)

Finiamo (N, e) e sia A tale che $A([x^e \bmod N]) = lsb(x)$

Dati N, e ed $y = [x^e \bmod N]$, possiamo calcolare i bit di x uno dopo l'altro, dal meno significativo al più significativo.

Infatti:

- per determinare $lsb(x)$, eseguiamo A .

Poi, procediamo come segue:

- sia $lsb(x) = 0$. Nota che $y/2^e = (x/2)^e \bmod N$ e, poiché x è pari, 2 divide x . Pertanto $x/2$ è lo shift a destra di un posto di x , e $lsb(x/2)$ è il secondo bit meno significativo di x .

Possiamo ottenerlo calcolando

$$y' = \left\lceil \frac{y}{2} \text{ mod } N \right\rceil \text{ e, poi, } A(y')$$

- sia $\text{lsb}(x) = 1$. In questo caso occorre notare che

$$\left\lceil \frac{x}{2} \text{ mod } N \right\rceil = (x + N)/2 \quad \left(\text{e.g. } \downarrow \quad s = \left\lceil \frac{3}{2} \text{ mod } 5 \right\rceil = \frac{(3+5)}{2} \right)$$

$2^{-1} \text{ mod } 5 = 3$

Quindi, $\text{lsb}(\lceil \frac{x}{2} \text{ mod } N \rceil)$ risulta uguale a $2 \text{sb}(x+N)$

↓
secondo bit meno
significativo

$$2 \text{sb}(x+N) = 1 \oplus 2 \text{sb}(N) \oplus 2 \text{sb}(x)$$

↓
(abbiamo un riporto perché sia
 x che N sono dispari)

$$\begin{array}{c} 1 \\ - \\ b \\ - \\ c \\ \hline \oplus \end{array}$$

Pertanto, se calcoliamo

$$y' = [y/e \bmod N], \text{ allora risulta}$$

$$\begin{aligned} zrb(x) &= A(y') \oplus 1 \oplus zrb(N) \\ &\quad \text{bb}\left(\frac{x+N}{2}\right) \end{aligned} \quad \hookrightarrow N \text{ pubblico}$$

Procedendo in modo similare possiamo calcolare tutti i bit di x .

A questo punto vediamo come usare l'bb per cifrare un bit in modo sicuro.

CONSTRUCTION 11.32

Let GenRSA be as usual, and define a public-key encryption scheme as follows:

- **Gen:** on input 1^n , run GenRSA(1^n) to obtain (N, e, d) . Output the public key $pk = \langle N, e \rangle$, and the private key $sk = \langle N, d \rangle$.
- **Enc:** on input a public key $pk = \langle N, e \rangle$ and a message $m \in \{0, 1\}$, choose a uniform $r \in \mathbb{Z}_N^*$ subject to the constraint that $\text{lsb}(r) = m$. Output the ciphertext $c := [r^e \bmod N]$.
- **Dec:** on input a private key $sk = \langle N, d \rangle$ and a ciphertext c , compute $r := [c^d \bmod N]$ and output $\text{lsb}(r)$.

Teorema. RSA difficile \Rightarrow Costruzione CPA-sicura
(ul. a ben RSA)

Dim. Mostriamo che Π ha cifrature indistinguibili in presenza di un Adv di ascolto $\Rightarrow \Pi$ è CPA-sicuro

Sia A ppt, $m_0 = 0$ ed $m_1 = 1$ in $\text{Pub}_{A,\Pi}^{\text{car}}(n)$.

$$\Pr_{\mathbf{c}} [\text{Pub}_{A,\Pi}^{\text{car}}(n) = 1] = \frac{1}{2} \Pr [A(N, e, c) = 0 \mid c \text{ è una cifratura di } m_0] \\ + \frac{1}{2} \Pr [A(N, e, c) = 1 \mid c \text{ è una cifratura di } m_1]$$

il bit b per
cifrare m_b è
scelto unif. a caso in

$$\text{Pub}_{A,\Pi}^{\text{car}}(n)$$

D'altra parte:

supponiamo di eseguire A nell'esperimento RSA-lsb :

$$\Pr_{\mathbf{c}} [\text{RSA-lsb}_{A, \text{Gen RSA}}(n) = 1] = \Pr [A(N, e, [2^e \bmod N]) = \text{lsb}(z)]$$

è scelto uniformemente a caso

Poiché $P_r [\text{lsb}(r) = 1] = 1/2$, risulta

$$P_r [\text{RSA-lsb}_{A, \text{benRSA}}(n) = 1] = \frac{1}{2} \cdot P_r [A(N, e, [2^e \bmod N]) = 0 \mid \text{lsb}(r) = 0] \\ + \frac{1}{2} P_r [A(N, e, [2^e \bmod N]) = 1 \mid \text{lsb}(r) = 1]$$

I due esperimenti coincidono. Il challenger in Pub sceglie m a caso.
Pertanto c è calcolata su una stringa uniforme. In RSA r è uniforme.

$$P_r [A(N, e, c) = b \mid c \text{ è una cifr. di } b] = P_r [A(N, e, [2^e \bmod N]) = b \mid \text{lsb}(r) = b]$$

$$\Rightarrow P_r [\text{Pub}_{A, \text{H}}^{ear}(n) = 1] = P_r [\text{RSA-lsb}_{A, \text{benRSA}}(n) = 1]$$

Avendo supposto che $\text{bb}(z)$ è un predicho hardcore per la permutazione RSA relativamente a GenRSA, $\exists \text{negl}(n)$:

$$\Pr_{\substack{A, \text{GenRSA}}} [\text{RSA-}\text{bb}_{A, \text{GenRSA}}(n) = 1] \leq \frac{1}{2} + \text{negl}(n)$$

$$\Rightarrow \Pr_{\substack{A, \Pi}} [\text{PubK}_{A, \Pi}^{\text{ear}}(n) = 1] \leq \frac{1}{2} + \text{negl}(n)$$

Pertanto Π è CPA-sicuro. □

KEM basato su cifratura con predotto hardcore

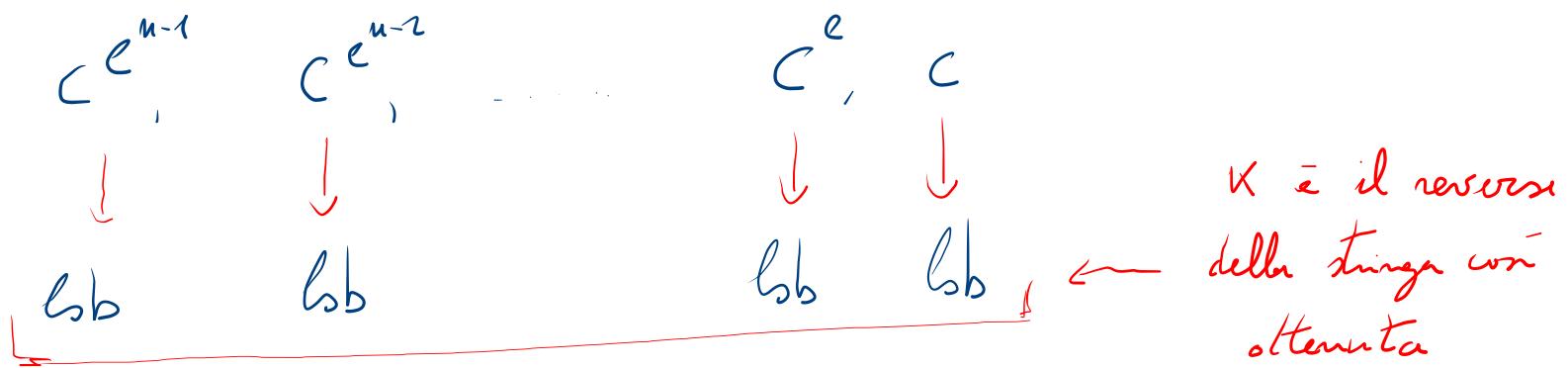
Soluzione immediata: scegli una chiave K di n bit
cifrali uno per uno

Problema: cifrato troppo lungo \rightarrow n elementi di \mathbb{Z}_N^*

Seconda soluzione: scegli c uniformemente a caso in \mathbb{Z}_N^*
(applicazione ripetuta della RSA)
calcola $c^e, c^{e^2}, c^{e^3}, \dots, c^{e^n}$ $\boxed{c^{e^n}}$ \leftarrow cifrato

$$K = \underbrace{\text{lsb}(c) \text{lbb}(c^e) \dots}_{n - \text{bit}} \text{lbb}(c^{e^{n-1}})$$

La chiave può essere recuperata decifrando c^{e^n} successivamente.



Ottene, calcolando $c = (c^{e^n})^{d^n}$ e poi procedendo
al ricalcolo delle cifrature e degli lsb (ordine giusto)



CONSTRUCTION 11.34

Let GenRSA be as usual, and define a KEM as follows:

- **Gen:** on input 1^n , run $\text{GenRSA}(1^n)$ to obtain (N, e, d) . Then compute $d' := [d^n \bmod \phi(N)]$ (note that $\phi(N)$ can be computed from (N, e, d) or obtained during the course of running GenRSA). Output $pk = \langle N, e \rangle$ and $sk = \langle N, d' \rangle$.
- **Encaps:** on input $pk = \langle N, e \rangle$ and 1^n , choose a uniform $c_1 \in \mathbb{Z}_N^*$. Then for $i = 1, \dots, n$ do:
 1. Compute $k_i := \text{lsb}(c_i)$.
 2. Compute $c_{i+1} := [c_i^e \bmod N]$.

Output the ciphertext c_{n+1} and the key $k = k_1 \dots k_n$.

- **Decaps:** on input $sk = \langle N, d' \rangle$ and a ciphertext c , compute $c_1 := [c^{d'} \bmod N]$. Then for $i = 1, \dots, n$ do:
 1. Compute $k_i := \text{lsb}(c_i)$.
 2. Compute $c_{i+1} := [c_i^e \bmod N]$.

Output the key $k = k_1 \dots k_n$.

Osservazione: la costruzione ricorda l'approccio usato per costruire un generatore pseudocasuale a partire da una permutazione one-way

$$\text{i.e., } f(x) = [x^e \bmod N] \quad (N, e)$$

pubbliche

Sicurezza CPA
della costruzione



pseudocasualità di
 $f^n(c) \# (f^{n-1}(c)) \dots \# (c)$

Teorema RSA difficile
rel. a SanRSA \Rightarrow KEM (CPA-sicuro)

Nota sull'efficienza:

- se $n = 128$ ed N è un intero di 2048 bit,
scgliendo $\underline{e=3}$ (2 moltiplicazioni modulari)

a) per cifrare occorrono $2 \cdot n = 256$ moltiplicazioni
modulari (produrre la capsula per la chiave K)

pili costosa
di una semplice
cifratura

b) per decifrare in media 3072 moltiplicazioni
modulari (un'esponentiazione piena) + 256
moltiplicazioni modulari

$$\left(\frac{256}{3072} \right) \times 100 \Rightarrow \text{solo l}'8\% \text{ pili costosa di una decifratura semplice}$$

Sicurezza CCA

Tutte le costruzioni basate su RSA viste fino ad ora non sono CCA-sicure. Plain RSA non è neanche CPA-sicuro.

Inoltre, Plain RSA è malleabile.

Data $c = m^e \text{ mod } N$, per qualche m , risulta:

$$c' = (2^e) \cdot c \text{ mod } N = (2 \cdot m)^e \text{ mod } N$$

$\Rightarrow c'$ è una cifratura di $2 \cdot m$ (relato ad m)

RSA PKCS #1 v1.5 è soggetto ad un attacco CCA importante proposto da Daniel Bleichenbach nel 1998.

Idea: dato un cifrato c che corrisponde ad una
cifratura corretta di un messaggio m (non noto)
rispetto ad (N, e)

Adv calcola $c' = [s^e \cdot c \bmod N]$ e lo invia al ricevente

Se $c = [\hat{m}^e \bmod N]$, $\hat{m} = 0x00 \parallel 0x02 \parallel 2 \parallel 0x00 \parallel m$

la decifratura di c' , darà:

$$\hat{m}' = s \cdot \hat{m} \bmod N$$

ed il ricevente darà "errore" se i primi 2 byte di

\hat{m}' non sono $0x00 \parallel 0x02$.

Si noti che
funziona come
un oracolo di
decifratura

Quando la decifrazione ha successo, Adv capisce che i primi due byte di $s \hat{=} m \bmod N$ sono $0x00 \text{ } 0x02$

\downarrow
(noto)

Operando in questo modo e sottraendosi soltanto dell'informazione decifrata / fallisce, Adv è in grado di ottenere un certo numero di equazioni modulari da cui riesce a ricavare m alla fine.

Nota: anche il KEM basato su RSA è CPA-sicuro, soggetto ad un attacco di tipo CCA.

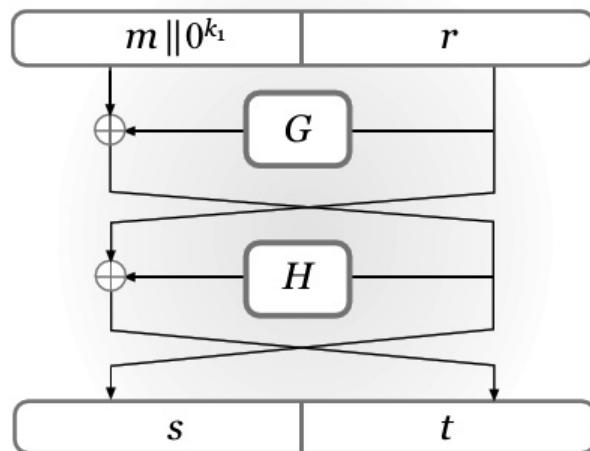
RSA OAEP

(Optimal asymmetric encryption padding)

Anche in questo caso basata su RSA con padding

$$m \xrightarrow[\text{randomizza}]{\text{trasformazione}} \hat{m} \in \mathbb{Z}_N^* \longrightarrow c = [\hat{m}^e \bmod N]$$

La trasformazione è però più complessa di prima:



Usa una rete di Feistel a due round
e due funzioni hash
← e il H

CONSTRUCTION 11.36

Let GenRSA be as in the previous sections, and ℓ, k_0, k_1 be as described in the text. Let $G : \{0, 1\}^{k_0} \rightarrow \{0, 1\}^{\ell+k_1}$ and $H : \{0, 1\}^{\ell+k_1} \rightarrow \{0, 1\}^{k_0}$ be functions. Construct a public-key encryption scheme as follows:

- **Gen:** on input 1^n , run $\text{GenRSA}(1^n)$ to obtain (N, e, d) . The public key is $\langle N, e \rangle$ and the private key is $\langle N, d \rangle$.
- **Enc:** on input a public key $\langle N, e \rangle$ and a message $m \in \{0, 1\}^\ell$, set $m' := m \| 0^{k_1}$ and choose a uniform $r \in \{0, 1\}^{k_0}$. Then compute

$$s := m' \oplus G(r), \quad t := r \oplus H(s)$$

and set $\hat{m} := s \| t$. Output the ciphertext $c := [\hat{m}^e \bmod N]$.

- **Dec:** on input a private key $\langle N, d \rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute $\hat{m} := [c^d \bmod N]$. If $\|\hat{m}\| > \ell + k_0 + k_1$, output \perp . Otherwise, parse \hat{m} as $s \| t$ with $s \in \{0, 1\}^{\ell+k_1}$ and $t \in \{0, 1\}^{k_0}$. Compute $r := H(s) \oplus t$ and $m' := G(r) \oplus s$. If the least-significant k_1 bits of m' are not all 0, output \perp . Otherwise, output the ℓ most-significant bits of \hat{m} .

Una versione di RSA OAEP fa parte dello standard

RSA PKCS #1 a partire dalla v2.0

La riduzione di sicurezza è piuttosto complicata ma si può mostrare che

BSA difficile + H-e 6 \Rightarrow RSA-OAEP
 rel. a GenRSA BOM CCA-sicura

Intuizione del risultato:

- durante la ciphatura il mittente calcola

$$m' = m \parallel 0^{kr}, \quad s = m' \oplus \underbrace{\ell(z)}_{\text{uniform}}, \quad t = z \oplus H(s)$$

Il ciprato risultante è

$$c = \left[\underbrace{(s \parallel t)}_m^{\hat{m}} \right]^e \text{mod } N$$

Se Adv non chiede r all oracolo f , $f(r)$ è uniforme per Adv e, quindi, m è protetto come nel ONE-TIME PAD in s .

Potrebbe Adv inviare la query " r " a f ?

Nota che il valore di r è protetto da $H(s)$ in t . Quindi se Adv NON chiede s all oracolo H , ancora r è protetto come nel ONE-TIME PAD in t .

Inoltre, se la lunghezza K_0 di r è suff. grande, la probabilità

che Adv indovini 2 tentando valori casuali è trascurabile (i.e., esponenzialmente piccola)

Pertanto, l'unica cosa che Adv può fare è

calcolare s da $\left[(s \parallel t)^e \bmod N \right]$

può poter poi inviare la query "s" all'oracolo H .

Attenzione: calcolare s da $\left[(s \parallel t)^e \bmod N \right]$ NON SIGNIFICA inserire le permutazioni RSA !

Nonostante ciò, si può dimostrare che:

- il recupero di s tramite A^{ppt}

\Rightarrow il recupero di t in poly-time

\Rightarrow l'inversione della permutazione RSA
in poly-time

Pertanto, recuperare s è computazionalmente inammissibile.

Nota: nel 2001, James Manger propose un attacco CCA contro
una implementazione de restitura due tipi diversi di emoji
quando la decifratura fallisce, invece dell'unico \perp predefinito

Attenzione, quindi, ad implementare correttamente.

Un KEM CCA-sicuro basato su RSA
La costruzione è semplice ed elegante.
La riduzione di sicurezza non è complicata.

CONSTRUCTION 11.37

Let GenRSA be as usual, and construct a KEM as follows:

- **Gen:** on input 1^n , run GenRSA(1^n) to compute (N, e, d) . The public key is $\langle N, e \rangle$, and the private key is $\langle N, d \rangle$.
As part of key generation, a function $H : \mathbb{Z}_N^* \rightarrow \{0, 1\}^n$ is specified, but we leave this implicit.
- **Encaps:** on input public key $\langle N, e \rangle$ and 1^n , choose a uniform $r \in \mathbb{Z}_N^*$. Output the ciphertext $c := [r^e \bmod N]$ and the key $k := H(r)$.
- **Decaps:** on input private key $\langle N, d \rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute $r := [c^d \bmod N]$ and output the key $k := H(r)$.

Teorema : RSA difficile + H \Rightarrow KEM CCA-secure
 w/ a GenRSA + ROM

Dim (Sketch) A ppt consideriamo $KEM_{A, \Pi}^{cca}(n)$

1. $\text{GenRSA}(1^n)$ is run to obtain (N, e, d) . In addition, a random function $H : \mathbb{Z}_N^* \rightarrow \{0, 1\}^n$ is chosen.
2. Uniform $r \in \mathbb{Z}_N^*$ is chosen, and the ciphertext $c := [r^e \bmod N]$ and key $k := H(r)$ are computed.
3. A uniform bit $b \in \{0, 1\}$ is chosen. If $b = 0$ set $\hat{k} := k$. If $b = 1$ then choose a uniform $\hat{k} \in \{0, 1\}^n$.
4. \mathcal{A} is given $pk = \langle N, e \rangle$, c , and \hat{k} , and may query $H(\cdot)$ (on any input) and the decapsulation oracle $\text{Decaps}_{\langle N, d \rangle}(\cdot)$ on any ciphertext $\hat{c} \neq c$.
5. \mathcal{A} outputs a bit b' . The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

Sia Query l'evento che, ad un certo punto, A fa la query "2" all'oracolo H . E sia Success l'evento $b = b'$.

$$\Pr_2[\text{Success}] = \Pr_2[\text{Success} \wedge \overline{\text{Query}}] + \Pr_2[\text{Success} \wedge \text{Query}]$$

$$\leq \Pr_2[\text{Success} \wedge \overline{\text{Query}}] + \Pr_2[\text{Query}]$$

(tutte le probabilità sono calcolate nelle scelte casuali in $\text{KEM}_{A, \Pi}^{\text{cca}}(n)$)

Mostriamo che: $\leq 1/2$ $\text{negl}(n)$

$$\Pr_2[\text{Success} \wedge \overline{\text{Query}}] = \Pr[\overline{\text{Query}}] \cdot \Pr_2[\text{Success} \mid \overline{\text{Query}}]$$

$\Pr_2[\text{Success} \wedge \overline{\text{Query}}] = 0 \leq 1/2$

$$\Pr_2[\text{Success} \wedge \overline{\text{Query}}] \leq \Pr_2[\text{Success} \mid \overline{\text{Query}}]$$

Condizionato a Query, il valore $K = H(z)$ è unif. distribuito
↳ è un oracolo casuale

Consideriamo $\text{KEM}_{A, \Pi}^{\text{cca}}(n)$.

Sia chiave pubblica pk e c determiniamo univocamente r
ma, poiché H è scelta indipendentemente da ogni altra, NON
danno alcuna info su $H(r)$.

Le query di A ad H non risiedono info su $H(r)$

↳ ancora perciò H è un oracolo casuale

Le query a Decaps_(N, d)(\hat{c}) = $H(\hat{r})$, dove $\hat{r} = [\hat{c}^d \bmod N]$,

non danno info su $H(r)$

↳ sempre perciò H è un oracolo casuale

(RSA è una perm.)
 $\hat{c} \neq c$
 $\hat{r} \neq r$

Pertanto, fino a quando $\overline{\text{Query}}$ non si verifica, il valore di $K = t/(2)$ è uniforme agli occhi di A dato p_K^* , cioè è imposto a tutte le query agli oracoli H e Decaps .

$$\Rightarrow \Pr[\text{Success} \mid \overline{\text{Query}}] = 1/2.$$

Nota: non abbiamo mai usato l'ipotesi A ppt.

La stessa prova vale se A è illimitato.

Dobbiamo mostrare che $\Pr[\text{Query}] \leq \text{negl}(n)$

Idea: usiamo A per costruire A' da, riceve (N, e, c) , istanza del problema BSA, e calcola z tale che

$$2^e \bmod N = c$$

Per far ciò, A' esegue A e

- risponde alle query di A per H (facile)
- risponde alle query di A per Decaps (più complicato)
A non dispone di SK

tuttavia A' può rispondere inviando valori casuali
infatti, rebbero Decaps (\hat{c})

- prima calcola \hat{z} : $\hat{z}^e = \hat{c} \bmod N$

- poi calcola $H(\hat{z})$
 $\xrightarrow{\text{è un orolo casuale}}$

il risultato finale è sempre una stringa uniforme

A' deve però fare attenzione a rispondere consistentemente
alle query per H e Decaps

Condizione di costanza:

- per ogni \hat{z}, \hat{c} con $\hat{z}^e = \hat{c} \bmod N$, risulta

$$H(\hat{z}) = \text{Decaps}(\hat{c})$$

A' può garantire usando due liste

L_H = risposte alle query per H

L_{Decaps} = risposte alle query per Decaps

e rispondendo opportunamente (dettagli seguono)

\mathcal{A}' simula KEM^{cca}(n) per \mathcal{A}

Algorithm \mathcal{A}' :

The algorithm is given (N, e, c) as input.

1. Initialize empty lists L_H, L_{Decaps} . choose a uniform $k \in \{0, 1\}^n$ and store (c, k) in L_{Decaps} .
2. Choose a uniform bit $b \in \{0, 1\}$. If $b = 0$ set $\hat{k} := k$. If $b = 1$ then choose a uniform $\hat{k} \in \{0, 1\}^n$. Run \mathcal{A} on $\langle N, e \rangle, c$, and \hat{k} .

When \mathcal{A} makes a query $H(\tilde{r})$, answer it as follows:

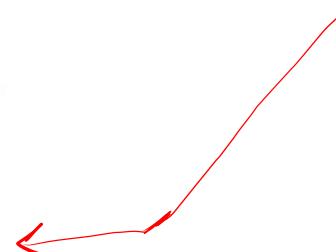
- If there is an entry in L_H of the form (\tilde{r}, k) for some k , return k .
- Otherwise, let $\tilde{c} := [\tilde{r}^e \bmod N]$. If there is an entry in L_{Decaps} of the form (\tilde{c}, k) for some k , return k and store (\tilde{r}, k) in L_H .
- Otherwise, choose a uniform $k \in \{0, 1\}^n$, return k , and store (\tilde{r}, k) in L_H .

nuovo \rightarrow

When \mathcal{A} makes a query $\text{Decaps}(\tilde{c})$, answer it as follows:

- If there is an entry in L_{Decaps} of the form (\tilde{c}, k) for some k , return k .
- Otherwise, for each entry $(\tilde{r}, k) \in L_H$, check if $\tilde{r}^e = \tilde{c} \bmod N$ and, if so, output k .
- Otherwise, choose a uniform $k \in \{0, 1\}^n$, return k , and store (\tilde{c}, k) in L_{Decaps} .

3. At the end of \mathcal{A}' 's execution, if there is an entry (r, k) in L_H for which $r^e = c \bmod N$ then return r .



query \tilde{r} per H

x c'è un (\tilde{r}, k) in L_H
tale che $\tilde{r}^e = \tilde{r} \bmod N$, restituisce K

query \tilde{c} per Decaps

x c'è un (\tilde{c}, k) in L_{Decaps}
tale che $\tilde{c}^e = \tilde{c} \bmod N$, restituisce K

tentativo di inversione

della permutazione RSA

Operazioni

- A' è ppt $\times A$ è ppt
- la vista di A
subroutine di A' \equiv la vista di A in
 $KEM_{A, \Pi}^{cca}(n)$
identiche
- A' dà in output la soluzione corretta quando
Query si verifica.

Pertanto :

$$\text{RSA difficile} \Rightarrow \Pr[\text{Query}] = \text{negl}(n)$$

rel. a ten RSA

Aspetti implementativi

È possibile migliorare l'efficienza delle implementazioni usando il teorema cinese del resto

Sia $N = p \cdot q$. Il ricavante vuole calcolare l'e-ima radice di y usando $d = e^{-1} \pmod{\varphi(N)}$. Poiché

$$y^d \pmod{N} \longleftrightarrow (y^d \pmod{p}, y^d \pmod{q})$$

può calcolare:

$$x_p = [y^d \pmod{p}] = [y^{d \pmod{p-1}} \pmod{p}]$$

$$x_q = [y^d \pmod{q}] = [y^{d \pmod{q-1}} \pmod{q}]$$

e poi $x \longleftrightarrow (x_p, x_q)$ (... lesson teoria dei numeri)

Il tempo di calcolo si riduce di circa $1/\frac{4}{3}$.

Inoltre: $[d \bmod (p-1)]$ e $[d \bmod (q-1)]$

possono essere precomputati.

Attenzione: una cattiva implementazione di usa
il CRT può "rilevare" i fattori p e q !

Per esempio: supponiamo che $[y^d \bmod N]$ venga
calcolato due volte.

- La prima correttamente, dando x

- La seconda con un errore, dando x'

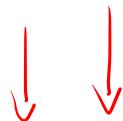
Supponiamo che molti

$$x' \equiv x \pmod{p} \quad \text{e} \quad x' \not\equiv x \pmod{q}$$

(e' enore \downarrow nel calcolo di x_q)

Ma allora : $p \mid (x' - x)$ ma $q \nmid (x' - x)$

Pertanto , $\gcd(x' - x, N) = p$



Adr disporre delle due decifrazione
riesce a fattorizzare N !

Nelle implementazioni hardware il ricalcolo è comune .

Le chiavi pubbliche degli utenti devono essere indipendenti

- Supponiamo che il modulo N sia lo stesso per diversi utenti
che hanno coppie (e_i, d_i)

Ma sappiamo che, dati N, e, d tali che $e \cdot d \equiv 1 \pmod{\varphi(n)}$

è facile fattorizzare N (- vedi lezioni teoria dei numeri)

\Rightarrow dati p e q calcolare i valori d_i degli
altri utenti è immediato

- Supponiamo che gli utenti si fidino l'uno dell'altro
e supponiamo che lo stesso messaggio venga inviato a 2 di essi

$$c_1 = m^{e_1} \bmod N$$

$$c_2 = m^{e_2} \bmod N$$

Consideriamo il caso in cui (N, e_1) ed (N, e_2)
siano tali che $\gcd(e_1, e_2) = 1$

di sì

di sì

$$\Rightarrow \exists x, y \text{ tali che } xe_1 + ye_2 = 1$$

(... lezioni di teoria dei numeri)

Pertanto, un AdL può calcolare -

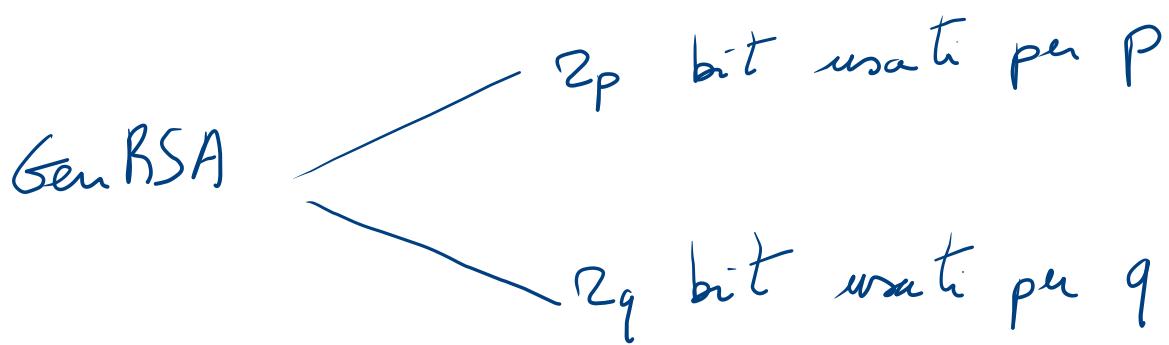
$$c_1^x c_2^y = m^{xe_1} \cdot m^{ye_2} = m^{xe_1 + ye_2} = m^1 = m \bmod N$$

La qualità della randomness deve essere buona in GenRSA

Occorre essere certi di un valore, invece di esser scelto uniformemente in $\{0,1\}^e$, non sia scelto in $S \subset \{0,1\}^e$ molto più piccolo.

\Rightarrow ricerche esauritive sono possibili.

In alcuni casi può andare anche peggio



Assumiamo di avere chiavi usino la stessa sorgente di random bit di bassa qualità, scelti unif. da un insieme di taglia 2^S .

Dopo aver generato 2^{Sk} chiavi (pk, sk) ci aspettiamo due moduli differenti, N ed N' , generati dalla stessa randomness, i.e., $\mathbb{Z}_p = \mathbb{Z}_{p'}$

$\Rightarrow N$ ed N' condividono un fattore primo comune

$$\Rightarrow \gcd(N, N') = p$$

Adv può cercare chiavi pubbliche su Internet e tentare (... realmente accaduto)

Descrizioni alternative di RSA

$$\underline{m \in \mathbb{Z}_N}$$

$$N = p \cdot q \quad (\text{primi dispari grandi})$$

$$e, d \text{ tali che } e \cdot d \equiv 1 \pmod{\varphi(N)}$$

Cifratina

$$c = m^e \pmod{N}$$

Decifratina

$$m = c^d \pmod{N}$$

$$m \in \mathbb{Z}_N$$

invece di a \mathbb{Z}_N^*

Perché funziona ?

Lemma $c^d \equiv m \pmod{N}$ se e solo se $c^d \equiv m \pmod{p}$ e $c^d \equiv m \pmod{q}$

$$\begin{aligned}
 c = m^e & \quad c^{ed} = (m^e)^d \pmod{p} & e \cdot d \equiv 1 \pmod{\varphi(N)} \\
 m \neq 0 & \quad = m^{K(p-1)(q-1) + 1} \pmod{p} & \Rightarrow e \cdot d = K\varphi(N) + 1 \\
 & \quad = m \cdot m^{K(p-1)(q-1)} \pmod{p} & = K(p-1)(q-1) + 1 \\
 & \quad = m \cdot (m^{(p-1)})^{K(q-1)} \pmod{p} & \xrightarrow{\text{Th. Fermat}}
 \end{aligned}$$

$$\begin{aligned}
 & = m \cdot (1)^{K(q-1)} \pmod{p} \\
 & = m \quad (\dots \text{pari identici per mod } q)
 \end{aligned}$$

$m = 0 \pmod{p}$, m multiplo di $p \Rightarrow m^{ed}$ multiplo di $p \Rightarrow m^{ed} = 0 \pmod{p}$

Altra descrizione

Funzione di Carmichael.

intero positivo n \longrightarrow intero positivo $\lambda(n)$

$\lambda(n) =$ il più piccolo intero positivo m tale che

$$a^m \equiv 1 \pmod{n}$$

\forall intero positivo $a \in \mathbb{Z}_n^*$

$\lambda(n) =$ esponente del gruppo moltiplicativo degli interi mod n

Nota: se $n = p$ (primo) $\lambda(p) = \varphi(p) = p - 1$

(Il gruppo moltiplicativo \mathbb{Z}_p^* è ciclico)

per n generico, $\lambda(n) \mid \varphi(n)$

Esempio $\lambda(8) = 2$ $a^2 \equiv 1 \pmod{8}$ $a \in \{1, 3, 5, 7\}$

$$1^2 \equiv 1 \pmod{8}, \quad 3^2 \equiv 1 \pmod{8}, \quad 5^2 \equiv 1 \pmod{8}, \quad 7^2 \equiv 1 \pmod{8}$$

$$\varphi(8) = 4 \quad (\# \text{ elementi coprimi con } 8)$$

Th Euler $\Rightarrow a^4 \equiv 1 \pmod{8}, \quad a \in \{1, 3, 5, 7\}$

ma 4 non è l'esponente più piccolo per cui accade.

Il **teorema di Carmichael** indica come calcolare $\lambda(n)$ se $n = p^k$, con p primo e k intero positivo:

$$\lambda(p^k) = \begin{cases} \frac{1}{2}\varphi(p^k) & \text{se } p = 2 \text{ e } k \geq 3, \\ \varphi(p^k) & \text{altrimenti,} \end{cases}$$

dove φ è la **funzione ϕ di Eulero** che per una potenza di un primo è data da:

$$\varphi(p^k) = p^{k-1}(p - 1).$$

Sappiamo come calcolare $\lambda(n)$.

$$n = p_1^{e_1} p_2^{e_2} \cdots p_n^{e_n} \Rightarrow \lambda(n) = \text{lcm}(\lambda(p_1^{e_1}), \dots, \lambda(p_n^{e_n}))$$

\hookrightarrow mcm (minimo comune
multiple)

Nel caso ch. $n = p \cdot q$ (p, q primi)

$$\lambda(n) = \text{lcm}(\lambda(p), \lambda(q)) = \underline{\underline{\text{lcm}((p-1), (q-1))}}$$

e sappiamo

$$\text{lcm}(a, b) = \frac{|a \cdot b|}{\text{gcd}(a, b)}.$$

Descrizione di GenRSA

1. Scegli p, q , primi distinti di n bit
2. Calcola $N = p \cdot q$
3. Calcola $\lambda(n) = \text{lcm}((p-1), (q-1))$
4. Scegli $1 < e < \underline{\lambda(n)}$ tale che
 $\text{gcd}(e, \underline{\lambda(n)}) = 1$
5. Calcola $d = e^{-1} \bmod \underline{\lambda(n)}$

Vantaggi: calcolare $d \bmod \varphi(n)$ invece che $\bmod \lambda(n)$

può a volte fornire valori più grandi del necessario

$$\text{i.e., } d > \lambda(n)$$