Schemi basati sull'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

$(N, p, q) \leftarrow \mathsf{GenModulus}(1^n)$
$\phi(N) := (p-1)(q-1)$
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 $\text{GenRSA}(1^n)$ 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
$$m := [c^d \bmod N].$$

Esempio giocattolo

$(N, e, d) \longleftarrow$ GenRSA( )

$(391, 3, 235)$

$\downarrow$

$17 \cdot 23 \rightarrow \varphi(391) = 16 \cdot 22 = 352 \rightarrow 3 \cdot 235 \equiv 1 \bmod 352$

$p \quad q$

$PK = \langle 391, 3 \rangle \qquad SK = \langle 391, 235 \rangle$

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

$$c = [158^3 \bmod 391] = 295$$

Per decifrare

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

Circa la sicurezza ...

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 ed e è computazionalmente inammissibile calcolare m.

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

Per capirci ...

- se il msg m non è scelto uniformemente in $\mathbb{Z}_n^*$?
- se Adv 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 subexp time
- algoritmi che sfruttano "e" piccolo
- algoritmi che sfruttano informazioni parziali su m
- algoritmi che sfruttano messaggi related o cifrature multiple di m

RSA randomizzato

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

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

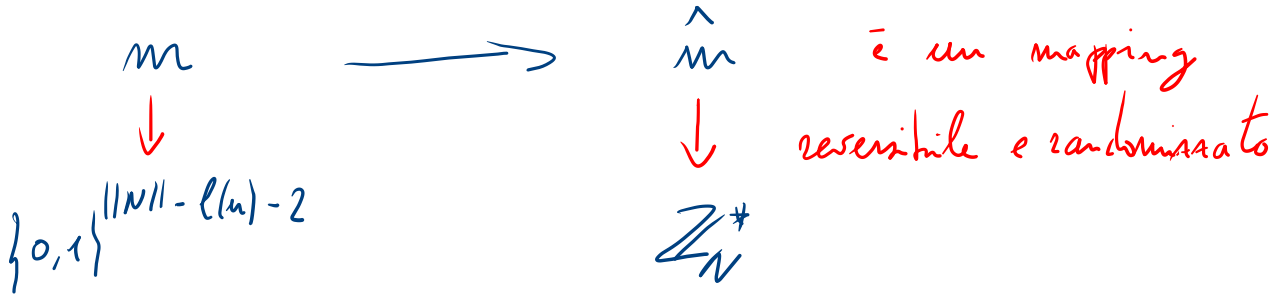
- Dec: on input a private key $sk = \langle N, d \rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute

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

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

The padded RSA encryption scheme.

Osservazione:



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

La sicurezza di Padded RSA dipende da ℓ.

Se ℓ è troppo piccolo, possono essere applicati tutti i possibili valori di r ed applicato l'attacco che sfrutta 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 nè
esistere se vale l'assunzione RSA,
momento non se ne conoscono.

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

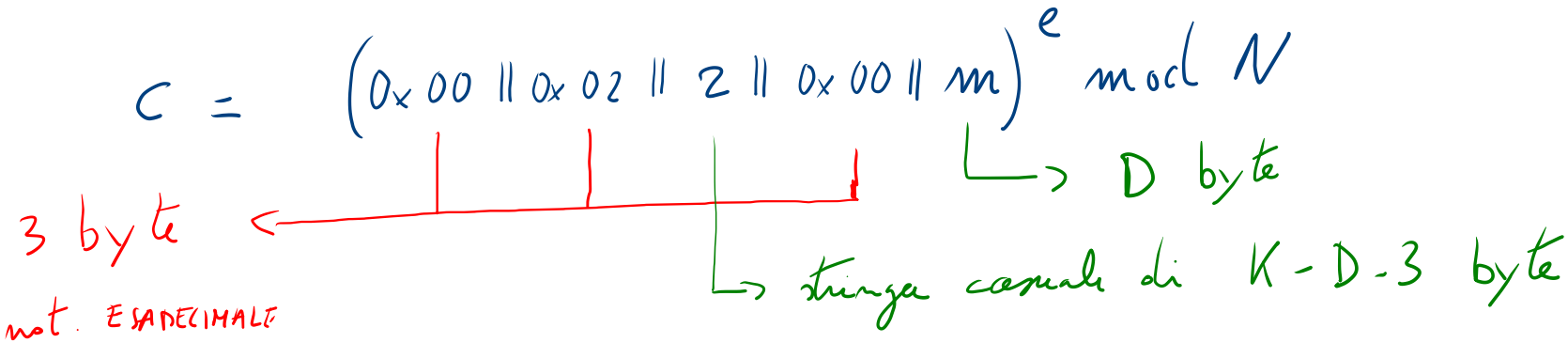
- K, lunghezza di N in byte, vale a dire

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

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

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

La cifratura di un messaggio m allora diviene:



Poiché $D \leq K - 11 \Rightarrow 2$ è di ALMENO 8 byte

Purtroppo, non è CPA-sicuro!

Un Adv può calcolare la parte iniziale di un msg m di cui si sa che ha molti 0 alla fine

Per capire, sia $m = b \| \underbrace{0 \ldots 0}_{L}$, $b \in \{0,1\}$ NON NOTO

di lunghezza massima, i.e., $L = 8 \cdot (k-11) - 1$. Risulta

$$c = (0x00 \| 0x02 \| 2 \| 0x00 \| b \| 0 \ldots 0)^e \bmod N$$

Adv può calcolare $c' = c / (2^L)^e \bmod N$. Risulta:

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

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

$2 + 64 + 8 + 1$

Pertanto, Adv applicando

- gli attacchi per messaggi brevi, oppure
- gli attacchi basati su conoscenza parziale di m

riesce a calcolare b

Occorre, quindi, avere valori di 2 "più grandi".

Se 2 è circa la metà del messaggio si 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 $\mathcal{A}$,
definiamo l'esperimento $\mathrm{RSA\_lsb}_{\mathcal{A},\mathrm{GenRSA}}(n)$

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

1. *Run* $\mathsf{GenRSA}(1^n)$ *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* $\mathsf{lsb}(x) = b$.

↓ Adv vince

↓ bit meno significativo

Nota: $\mathrm{lsb}(x)$ è un bit uniforme quando $x \in \mathbb{Z}_N^*$ viene scelto uniformemente a caso.

$\mathcal{A}$ può indovinare $\mathrm{lsb}(x)$ con prob. $1/2$.

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

$\exists$ negl(n) tale che

$$P_2\left[\mathrm{RSA\_lsb}_{A,\mathrm{GenRSA}}(n) = 1\right] \leq \frac{1}{2} + \mathrm{negl}(n)$$

In altre parole, lsb(x) è un predicato hardcore per la permutazione RSA.

Intuizione sulla prova del teorema.

Se esistesse un alg. efficiente che calcola SEMPRE lsb(z) da N, e ed $z^e \bmod N$, allora potrebbe essere usato per calcolare efficientemente x da N, e ed $x^e \bmod N$

(... invertire RSA)

Fissiamo (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

$$y' = [y/2 \bmod N] \quad \text{e, poi, } A(y')$$

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

$$[x/2 \bmod N] = (x + N)/2 \quad \left(\text{e.g. } 4 = [3/2 \bmod 5] = \frac{(3+5)}{2}\right)$$

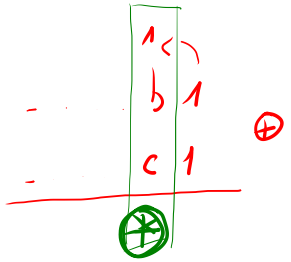
$2^{-1} \bmod 5 = 3$

Quindi, $lsb([x/2 \bmod N])$ risulta uguale a $2sb(x+N)$

↓ secondo bit meno significativo

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

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



Pertanto, se calcoliamo

$y' = [\, y/2^e \bmod N \,]$, allora risulta

$$lsb(x) = A(y') \oplus 1 \oplus lsb(N)$$

↳ N pubblico

$lsb\left(\frac{x+N}{2}\right)$

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

A questo punto vediamo come usare l'lsb 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 $\mathsf{lsb}(r) = m$. 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 $\mathsf{lsb}(r)$.

Teorema. RSA difficile $\Rightarrow$ Costruzione CPA-sicura (rel. a GenRSA)

CONSTRUCTION 11.32

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

- Gen: on input 1^n, run $\mathsf{GenRSA}(1^n)$ 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 $\mathsf{lsb}(r) = m$. 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 $\mathsf{lsb}(r)$.

Teorema. RSA difficile $\Rightarrow$ Costruzione CPA-sicura (rel. a GenRSA)

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

Sia A ppt, $m_0 = \emptyset$ ed $m_1 = 1$ in $Priv^{eav}_{A,\Pi}(n)$.

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

il bit b per cifrare m_b è scelto uniformemente a caso in $Priv^{eav}_{A,\Pi}(n)$

D'altra parte:

supponiamo di eseguire A nell'esperimento RSA_lsb:

$$Pr\left[RSA\_lsb_{A,GenRSA}(n) = 1\right] = Pr\left[A(N,e,[2^e \bmod N]) = lsb(2)\right]$$

2 scelto uniformemente a caso

Poiché $Pr[lsb(r) = 1] = 1/2$, risulta

$$Pr[RSA\text{-}lsb_{A, GenRSA}(n) = 1] = 1/2 \cdot Pr[A(N, e, [2^e \bmod N]) = 0 \mid lsb(r) = 0] + 1/2 \, Pr[A(N, e, [2^e \bmod N]) = 1 \mid lsb(r) = 1]$$

Notando che, cifrare $m \in \{0,1\}$, corrisponde a scegliere un r uniforme che soddisfa $lsb(r) = m$, discende che:

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

$$\Rightarrow Pr[Pub^{ear}_{A, \Pi}(n) = 1] = Pr[RSA\text{-}lsb_{A, GenRSA}(n) = 1]$$

Avendo supposto che $lsb(x)$ è un predicato hard-core per la permutazione RSA relativamente a GenRSA, $\exists\, negl(n)$:

$$P_r\left[RSA\_lsb_{A, GenRSA}(n) = 1\right] \leq \frac{1}{2} + negl(n)$$

$$\Rightarrow \quad P_r\left[PubK^{eav}_{A,\Pi}(n) = 1\right] \leq \frac{1}{2} + negl(n)$$

Pertanto Π è CPA-sicuro. ∎

KEM basato su cifratura con predicato hard core

Soluzione immediata: scegli una chiave K di n bit

cifrali uno per uno

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

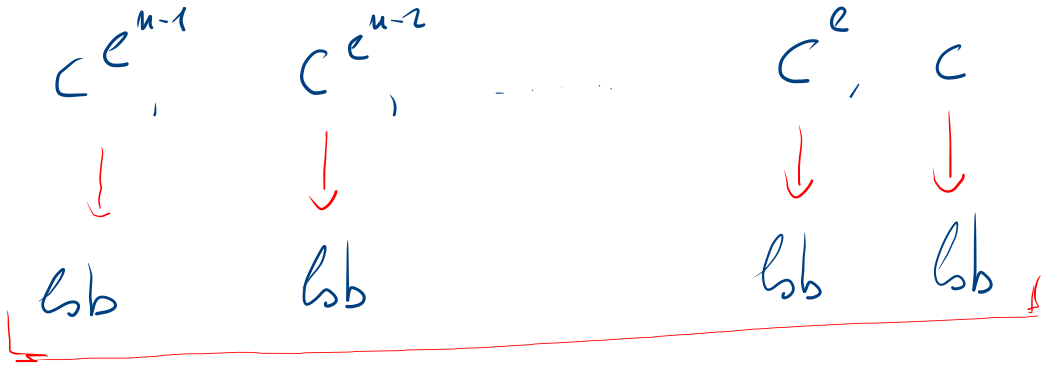
Seconda soluzione: scegli c uniformemente a caso in $\mathbb{Z}_N^*$

(applicazione ripetuta della RSA)

calcola $c^e, c^{e^2}, c^{e^3}, \ldots, \boxed{c^{e^n}}$ ← cifrato

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

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



K è il reverso della stringa così ottenuta

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

$c \rightarrow c^e \rightarrow \dots \rightarrow c^{e^{n-1}}$

$lsb(c)$ $lsb(c^e)$ $lsb(c^{e^{n-1}})$

CONSTRUCTION 11.34

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

- Gen: on input 1^n, run $\mathsf{GenRSA}(1^n)$ 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 $i = 1, \ldots, n$ do:

  1. Compute $k_i := \mathsf{lsb}(c_i)$.
  2. Compute $c_{i+1} := [c_i^e \bmod N]$.

  Output the ciphertext c_{n+1} and the key $k = k_1 \cdots k_n$.
- Decaps: on input $sk = \langle N, d' \rangle$ and a ciphertext c, compute $c_1 := [c^{d'} \bmod N]$. Then for $i = 1, \ldots, n$ do:

  1. Compute $k_i := \mathsf{lsb}(c_i)$.
  2. Compute $c_{i+1} := [c_i^e \bmod N]$.

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

Schemi basati sull'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

$(N, p, q) \leftarrow \mathsf{GenModulus}(1^n)$
$\phi(N) := (p-1)(q-1)$
choose $e > 1$ such that $\gcd(e, \phi(N)) = 1$
compute $d := [e^{-1} \bmod \phi(N)]$
return N, e, d

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

i.e., $f(x) = [x^e \bmod N]$ (N, e) pubbliche

Sicurezza CPA della costruzione $\Longleftrightarrow$ pseudocasualità di $f^u(c)\ lsb(f^{u-1}(c)) \ldots lsb(c)$

Teorema RSA difficile rel. a GenRSA $\Longrightarrow$ KEM CPA-sicuro

Nota sull'efficienza:

- se $n = 128$ ed N è un intero di 2048 bit,

scegliendo $e = 3$ (2 moltiplicazioni modulari)

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

più costosa di una semplice cifratura

b) per decifrare in media 3072 (2048 + 1024) moltiplicazioni modulari (un'esponenziazione piena) + 256 moltiplicazioni modulari

$\left(256/3072\right) \times 100$ => solo l'8% più 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 $c = m^e \bmod N$, per qualche m, risulta:

$$c' = (2^e) \cdot c \bmod N = (2 \cdot m)^e \bmod N$$

$\Rightarrow$ 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 Bleichenbacher 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 $c = [\hat{m}^e \bmod N]$, $\hat{m} = 0x00 \| 0x02 \| 2 \| 0x00 \| m$

la decifratura di c', darà:

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

Si noti che funziona come un oracolo di decifratura

ed il ricevente darà "errore" se i primi 2 byte di $\hat{m}'$ non sono $0x00 \| 0x02$.

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

(noto)

Operando in questo modo e servendosi soltanto dell'informazione decifra/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 che è CPA-sicuro, soccombe ad un attacco di tipo CCA.

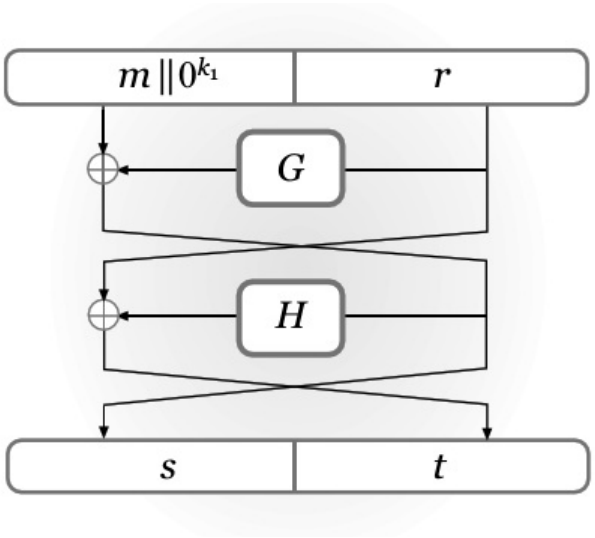
RSA OAEP

(Optimal asymmetric encryption padding)

Anche in questo caso basata su RSA con padding

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

La trasformazione è però più complessa di prima:



Usa una rete di Feistel a due round e due funzioni hash G ed 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 $G : \{0,1\}^{k_0} \to \{0,1\}^{\ell+k_1}$ and $H : \{0,1\}^{\ell+k_1} \to \{0,1\}^{k_0}$ be functions. Construct a public-key encryption scheme as follows:

- Gen: on input 1^n, run $\mathsf{GenRSA}(1^n)$ 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

RSA difficile rel. a GenRSA + H e G ROM $\Rightarrow$ RSA-OAEP CCA-sicura

Intuizione del risultato:

- durante la cifratura il mittente calcola

$$m' = m \| 0^{k_1}, \quad s = m' \oplus G(r), \quad t = r \oplus H(s)$$

$G(r)$: uniforme

Il cifrato risultante è

$$c = [(s \| t)^e \bmod N]$$

$\hat{m}$

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

Potrebbe Adv inviare la query "z" a G?

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

Inoltre, se la lunghezza k_0 di z è suff. grande, la probabilità

che Adv indovini a tentativo valori casuali è trascurabile (i.e., esponenzialmente piccola)

Pertanto, l'unica cosa che Adv può fare è

calcolare s da $[(s \| t)^e \bmod N]$

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

Attenzione: calcolare s da $[(s \| t)^e \bmod N]$ NON SIGNIFICA invertire la permutazione 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 BSA in poly-time.

Pertanto, recuperare s è computazionalmente insormontabile.

Nota: nel 2001, James Manger propose un attacco CCA contro una implementazione di restituzione due tipi diversi di errori 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 $H : \mathbb{Z}_N^* \to \{0,1\}^n$ 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 rel. a GenRSA + H BOM $\Rightarrow$ KEM CCA-sicuro

Dim (Sketch) $\mathcal{A}$ ppt. Consideriamo $\mathrm{KEM}^{cca}_{\mathcal{A},\Pi}(n)$

1. $\mathsf{GenRSA}(1^n)$ is run to obtain (N, e, d). In addition, a random function $H : \mathbb{Z}_N^* \to \{0,1\}^n$ 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 $\mathsf{Decaps}_{\langle N,d\rangle}(\cdot)$ 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 $b' = b$.

$$P_2[\text{Success}] = P_2[\text{Success} \wedge \overline{\text{Query}}] + P_2[\text{Success} \wedge \text{Query}]$$

$$\leq P_2[\text{Success} \wedge \overline{\text{Query}}] + P_2[\text{Query}]$$

(tutte le probabilità sono calcolate nelle stesse condizioni in $KEM^{cca}_{A,\pi}(n)$)

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

$$P_2[\text{Success} \wedge \overline{\text{Query}}] = P[\overline{\text{Query}}] \; P_2[\text{Success} \mid \overline{\text{Query}}]$$

$= 0$ segue immediatamente che $P_2[\text{Success} \wedge \overline{\text{Query}}] = 0 \leq 1/2$

> 0 $\quad P_2[\text{Success} \wedge \overline{\text{Query}}] \leq P_2[\text{Success} \mid \overline{\text{Query}}]$

Condizionato a ~~Query~~, il valore $K = H(r)$ è unif. distribuito

↳ è un oracolo casuale

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

La chiave pubblica pk e c determinano 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 rivelano info su $H(r)$

↳ ancora perché H è un oracolo casuale

La 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 perché H è un oracolo casuale

↳ $\hat{c} \neq c$

(RSA è una perm.) ⇓

$\hat{r} \neq r$

Pertanto, fino a quando $Query$ non si verifica, il valore di $K = H(z)$ è uniforme agli occhi di A dati pk, c e le risposte a tutte le query agli oracoli H e $Decaps$.

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

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

La stessa prova vale se A è illimitato.

Dobbiamo mostrare che $\Pr[Query] \leq negl(n)$.

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

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

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, sebbene Decaps ($\hat{c}$)

- prima calcola $\hat{z}$: $\hat{z}^e = \hat{c} \bmod N$
- poi calcola $H(\hat{z})$ → è un oracolo 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 consistenza:

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

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

A' può garantirla usando due liste

L_H = risposte alle query per H

L_{Decaps} = risposte alle query per Decaps

e rispondendo opportunamente (dettagli seguono)

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.

   When $\mathcal{A}$ makes a query $\mathsf{Decaps}(\tilde{c})$, 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.

A' simula $KEM^{cca}_{A,\Pi}(n)$ per A

query $\tilde{r}$ per H

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

query $\tilde{c}$ per Decaps

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

tentativo di inversione della permutazione RSA

Osservazioni

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

Pertanto:

RSA difficile $\Rightarrow$ $Pr[Query] \leq negl(n)$ rel. a fun RSA

Aspetti implementativi

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

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

$$y^d \bmod N \longleftrightarrow (y^d \bmod p,\ y^d \bmod q)$$

può calcolare:

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

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

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

Il tempo di calcolo si riduce di circa 1/4.

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

possono essere precomputati.

Attenzione: una cattiva implementazione del CRT può "rilasciare" 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 risulti

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

↓

(l'errore è nel calcolo di x_q)

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

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

↓ ↓

Adv disponendo delle due decifrature 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 $e \cdot d \equiv 1 \bmod \varphi(n)$

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

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

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

$$c_1 = m^{e_1} \bmod N \qquad c_2 = m^{e_2} \bmod N$$

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

(diversi)

$\Rightarrow \exists\, x, y$ tali che $x e_1 + y e_2 = 1$

(... lezioni di teoria dei numeri)

Pertanto, un Adv può calcolare ...

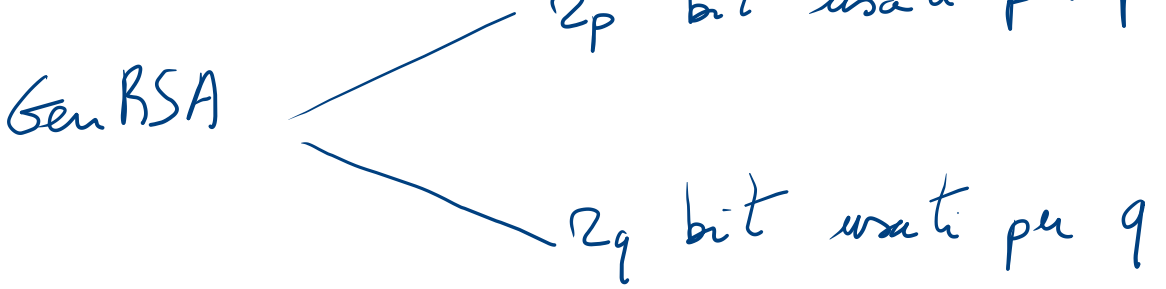
$$c_1^x \cdot c_2^y = m^{x e_1} \cdot m^{y e_2} = m^{x e_1 + y e_2} = m^1 = m \bmod N$$

La qualità della randomness deve essere buona in GenRSA

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

$\Rightarrow$ ricerche esaustive sono possibili.

In alcuni casi può andare anche peggio



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

Dopo aver generato $2^{s/2}$ chiavi (pk, sk) ci aspettiamo due moduli differenti, N ed N', generati dalla stessa randomness, i.e., $r_p = r'_p$

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

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

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