ECE297:11 Lecture 13

RSA – implementation issues & countermeasures against known attacks

Number of bits vs. number of decimal digits

$$10^{\#digits} = 2^{\#bits}$$

$$\#digits = (\log_{10} 2) \cdot \#bits \approx 0.30 \cdot \#bits$$

256 bits = 77 D
384 bits = 116 D
512 bits = 154 D
768 bits = 231 D
1024 bits = 308 D
2048 bits = 616 D

How to perform exponentiation efficiently?

$$Y = X^E \bmod N = \underbrace{X \cdot X \cdot X \cdot X \cdot X \ldots \cdot X \cdot X}_{\text{E-times}} \bmod N$$

E may be in the range of $2^{1024} \approx 10^{308}$

Problems:

1. huge storage necessary to store M^e before reduction
2. amount of computations infeasible to perform

Solutions:

1. modulo reduction after each multiplication
2. clever algorithms

200 BC, India, "Chandah-Sûtra"

Right-to-left binary exponentiation

$Y = X^E \bmod N$

$E = (e_{L-1}, e_{L-2}, \ldots, e_1, e_0)_2$

S: X $\quad X^2 \bmod N$ $\quad X^4 \bmod N$ $\quad X^8 \bmod N$ $\quad \ldots$ $\quad X^{2^{L-1}} \bmod N$

E: e_0 $\quad e_1$ $\quad e_2$ $\quad e_3$ $\quad \ldots$ $\quad e_{L-1}$

$$Y = X^{e_0} \cdot (X^2 \bmod N)^{e_1} \cdot (X^4 \bmod N)^{e_2} \cdot (X^8 \bmod N)^{e_3} \cdot \ldots \cdot (X^{2^{L-1}} \bmod N)^{e_{L-1}}$$

$(X^a)^b = X^{ab}$ $\qquad X^a \cdot X^b = X^{a+b}$

$$Y = X^{e_0 + 2\cdot e_1 + 4\cdot e_2 + 8\cdot e_3 + 2^{L-1}\cdot e_{L-1}} \bmod N =$$

$$= X^{\sum_{i=0}^{L-1} e_i \cdot 2^i} = X^E \bmod N$$

Right-to-left binary exponentiation: Example

$Y = 3^{19} \bmod 11$

$E = 19 = 16 + 2 + 1 = (10011)_2$

S:	X	$X^2 \bmod N$	$X^4 \bmod N$	$X^8 \bmod N$	$X^{16} \bmod N$
	3	$3^2 \bmod 11 = 9$	$9^2 \bmod 11 = 4$	$4^2 \bmod 11 = 5$	$5^2 \bmod 11 = 3$
E:	e_0	e_1	e_2	e_3	e_4
	1	1	0	0	1

$$Y = X \cdot X^2 \bmod N \cdot 1 \cdot 1 \cdot X^{16} \bmod N =$$

$$3 \cdot 9 \cdot 1 \cdot 1 \cdot 3 \bmod 11$$

$$= X^{19} \bmod N$$

$$(27 \bmod 11) \cdot 3 \bmod 11 = 5 \cdot 3 \bmod 11 = 4$$

Left-to-right binary exponentiation

$Y = X^E \bmod N$

$E = (e_{L-1}, e_{L-2}, \ldots, e_1, e_0)_2$

E:	e_{L-1}	e_{L-2}	e_{L-3}	…	e_1	e_0

$$Y = ((\ldots(((1^2 \cdot X^{e_{L-1}})^2 \cdot X^{e_{L-2}})^2 \cdot X^{e_{L-3}})^2 \quad \ldots.)^2 \cdot X^{e_1})^2 \cdot X^{e_0} \bmod N$$

$$(X^a)^b = X^{ab} \qquad X^a \cdot X^b = X^{a+b}$$

$$Y = X^{(e_{L-1} \cdot 2 + e_{L-2}) \cdot 2 + e_{L-3}) \cdot 2 + \ldots. + e_1) \cdot 2 + e_0} \bmod N =$$

$$= X^{2^{L-1} \cdot e_{L-1} + 2^{L-2} \cdot e_{L-2} + 2^{L-3} \cdot e_{L-3} + \ldots + 2 \cdot e_1 + e_0} \bmod N = X^{\sum_{i=0}^{L-1} e_i \cdot 2^i} =$$

$$= X^E \bmod N$$

Left-to-right binary exponentiation: Example

$Y = 3^{19} \bmod 11$

$E = 19 = 16 + 2 + 1 = (10011)_2$

E:	e_4	e_3	e_2	e_1	e_0
	1	0	0	1	1

$$Y = ((\ldots(((1^2 \cdot X)^2 \cdot 1)^2 \cdot 1)^2 \cdot X)^2 \cdot X \bmod N$$

$$= (((3^2 \bmod 11))^2 \bmod 11)^2 \bmod 11 \cdot 3)^2 \bmod 11 \cdot 3 \bmod 11$$

$$= (81 \bmod 11)^2 \bmod 11 \cdot 3)^2 \bmod 11 \cdot 3 \bmod 11 =$$

$$= (5 \cdot 3)^2 \bmod 11 \cdot 3 \bmod 11 =$$

$$= 4^2 \bmod 11 \cdot 3 \bmod 11 =$$

$$= 5 \cdot 3 \bmod 11 = \mathbf{4}$$

$$Y = (X^8 \cdot X)^2 \cdot X \bmod N = X^{19} \bmod N$$

Exponentiation: $Y = X^E \bmod N$

$E = (e_{L-1}, e_{L-2}, \ldots, e_1, e_0)_2$

Right-to-left binary exponentiation

```
Y = 1;
S = X;
for  i=0 to L-1
  {
    if (e_i == 1)
      Y = Y · S mod N;
    S = S^2 mod N;
  }
```

Left-to-right binary exponentiation

```
Y = 1;
for  i=L-1 downto 0
  {
    Y = Y^2 mod N;
    if (e_i == 1)
      Y = Y · X mod N;
  }
```

Exponentiation Example: $Y = 7^{12} \bmod 11$

Right-to-left binary exponentiation

Left-to-right binary exponentiation

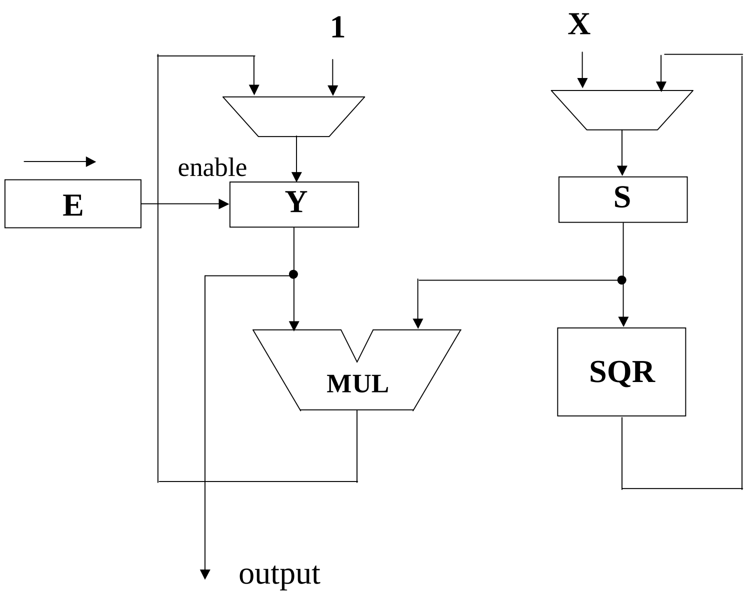
$12 = (1\ 1\ 0\ 0)_2$

i		**0**	**1**	**2**	**3**
$\mathbf{e_i}$		0	0	1	1
$\mathbf{S_{before}}$		7	5	3	9
$\mathbf{Y_{after}}$	1	1	1	3	**5**
$\mathbf{S_{after}}$	7	5	3	9	4

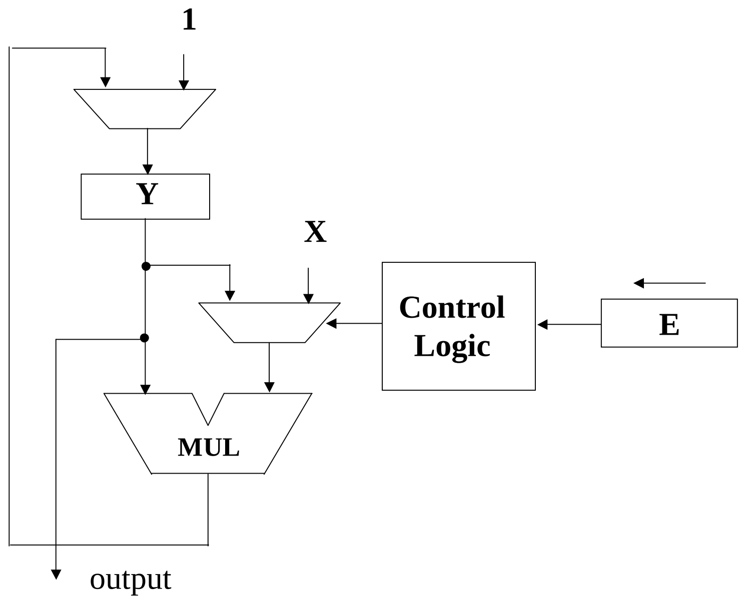
i		**3**	**2**	**1**	**0**
$\mathbf{e_i}$		1	1	0	0
Y	1	7	2	4	**5**

$\mathbf{S_{before}}$ - S before round i is computed
$\mathbf{S_{after}}$ - S after round i is computed

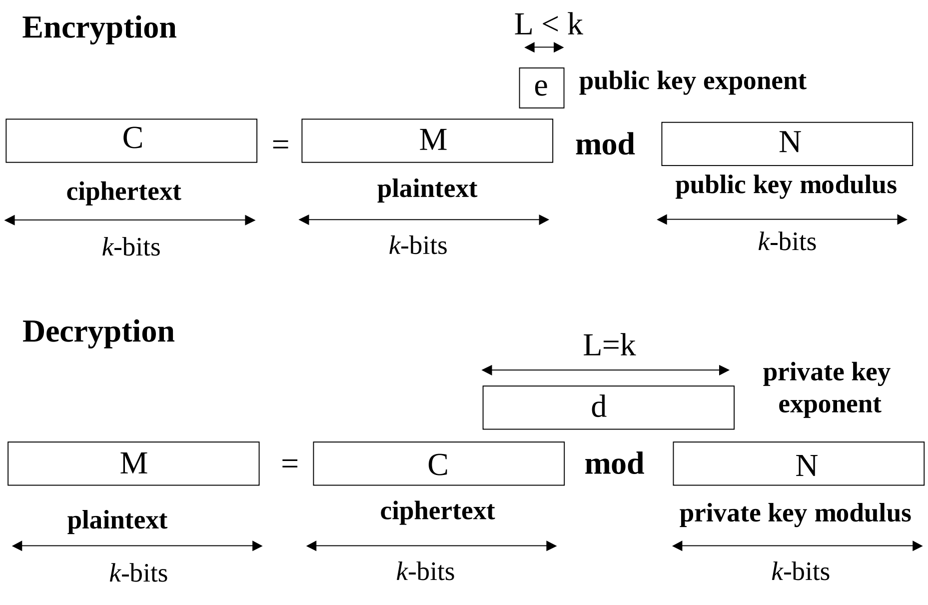
Right-to-Left Binary Exponentiation in Hardware



Left-to-Right Binary Exponentiation in Hardware



Basic Operations of RSA



Time of exponentiation

$t_{EXP}(e, L, k) = \#modular\_multiplications(e, L) \cdot t_{MULMOD}(k)$

e, L	#modular_multiplications
e=3	2
$e = F_4 = 2^{2^4} + 1$	17
large random L-bit e	$L + \#ones(1) \approx \frac{3}{2} \cdot L$

$t_{MULMOD}(k)$ - time of a single modular multiplication
of two k-bit numbers modulo a k-bit number

SOFTWARE

$t_{MULMOD}(k) = c_{sm} \cdot k^2$

HARDWARE

$t_{MULMOD}(k) = c_{hm} \cdot k$

Algorithms for Modular Multiplication

Multiplication

- Paper-and-pencil $\theta(k^2)$
- Karatsuba $\theta(k^{3/2})$
- Schönhage-Strassen (FFT) $\theta(k \cdot \ln(k))$

Modular Reduction

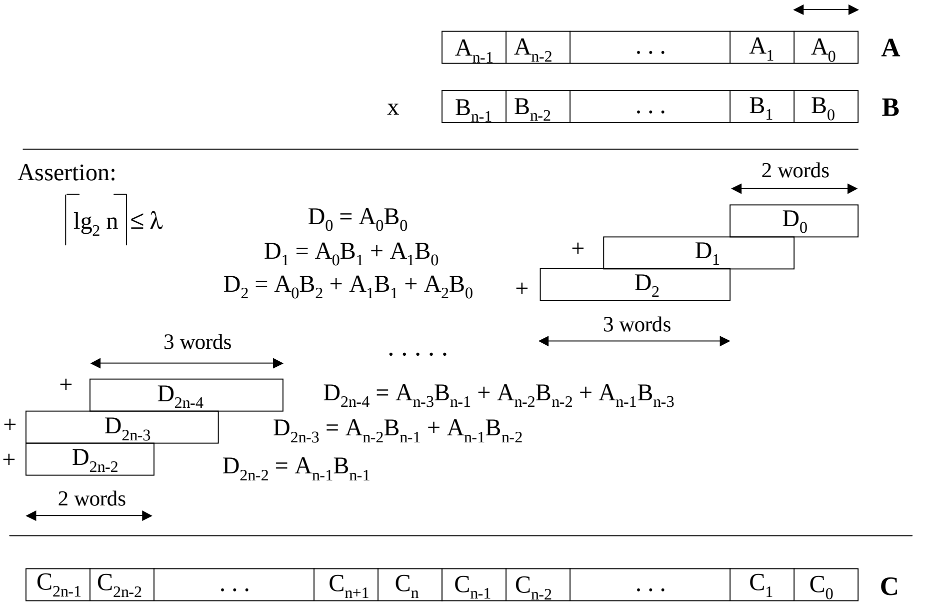
- classical $\theta(k^2)$
- Barrett complexity same as multiplication used
- Selby-Mitchell $\theta(k^2)$

Multiplication combined with modular reduction

- Montgomery algorithm

  $\theta(k^2)$

Paper-and-Pencil Algorithm of Multiplication



Assertion:

$\lceil \lg_2 n \rceil \le \lambda$

$D_0 = A_0B_0$

$D_1 = A_0B_1 + A_1B_0$

$D_2 = A_0B_2 + A_1B_1 + A_2B_0$

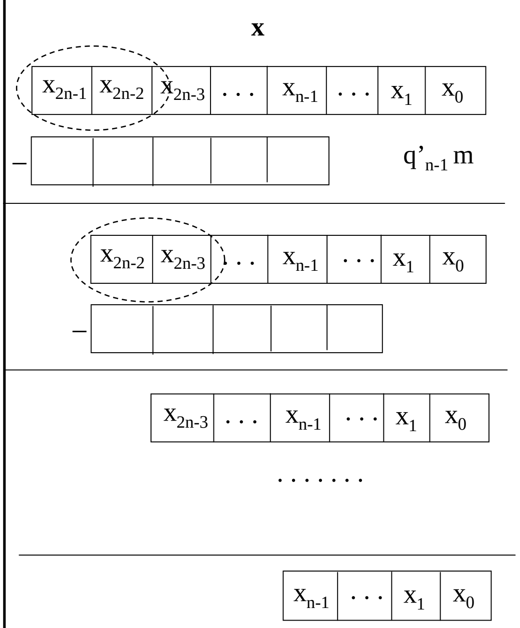
. . . . .

$D_{2n-4} = A_{n-3}B_{n-1} + A_{n-2}B_{n-2} + A_{n-1}B_{n-3}$

$D_{2n-3} = A_{n-2}B_{n-1} + A_{n-1}B_{n-2}$

$D_{2n-2} = A_{n-1}B_{n-1}$

Classical Algorithm (1)



$$q'_{n-1} = \left\lfloor \frac{x_{2n-1}b + x_{2n-2}}{m_{n-1}} \right\rfloor \qquad q'_{n-1} = q_{n-1} + \varepsilon$$

$\varepsilon = 0, 1, 2$

$$q'_{n-2} = \left\lfloor \frac{x_{2n-2}b + x_{2n-3}}{m_{n-1}} \right\rfloor \qquad q'_{n-2} = q_{n-2} + \varepsilon$$

$\varepsilon = 0, 1, 2$

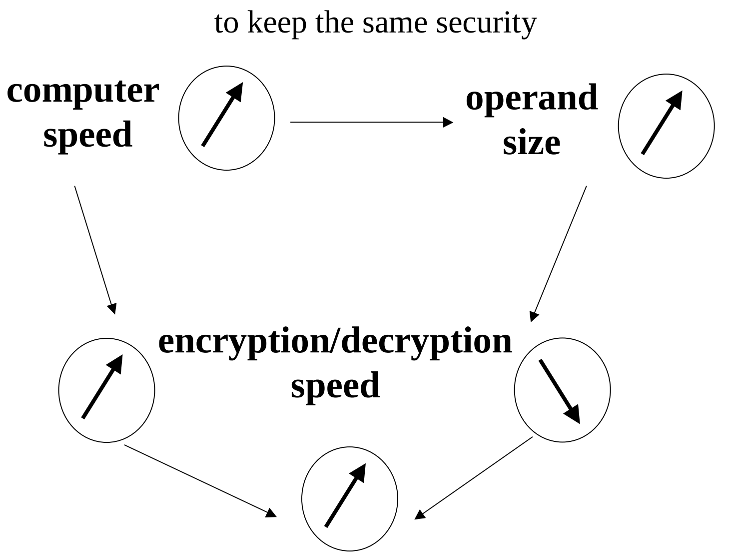
Time of basic operations in software and hardware

	SOFTWARE	HARDWARE
Modular Multiplication	$c_{sm} \cdot k^2$	$c_{hm} \cdot k$
Modular Exponentiation	$c_{sme} \cdot k^2 \cdot L$	$c_{hme} \cdot k \cdot L$

Time of the RSA operations as a function of the key size k

	SOFTWARE	HARDWARE
Encryption/ Signature verification with a small exponent e	$c_{se} \cdot k^2$	$c_{he} \cdot k$
Decryption / Signature generation	$c_{sd} \cdot k^3$	$c_{hd} \cdot k^2$
Key Generation	$c_{sk} \cdot k^4/\log_2 k$	$c_{hk} \cdot k^3/\log_2 k$
Factorization (breaking RSA)	$\exp(c_{sf} \cdot k^{1/3} \cdot (\ln k)^{2/3})$	

Effect of the increase in the computer speed on the speed of encryption and decryption in RSA



Decryption using Chinese Remainder Theorem

$$\boxed{M} = \boxed{C}^{\boxed{d}} \bmod \boxed{N}$$

$C_P = C \bmod P$
$d_P = d \bmod (P-1)$

$$\boxed{M_P} = \boxed{C_P}^{\boxed{d_P}} \bmod \boxed{P}$$

$C_Q = C \bmod Q$
$d_Q = d \bmod (Q-1)$

$$\boxed{M_Q} = \boxed{C_Q}^{\boxed{d_Q}} \bmod \boxed{Q}$$

$$M = M_P \cdot R_Q + M_Q \cdot R_P \bmod N$$

where

$R_P = (P^{-1} \bmod Q) \cdot P = P^{Q-1} \bmod N$

$R_Q = (Q^{-1} \bmod P) \cdot Q = Q^{P-1} \bmod N$

Time of decryption without and with Chinese Remainder Theorem

SOFTWARE

Without CRT

$$t_{DEC}(k) = t_{EXP}(\text{random } e, k, L{=}k) = c_s \cdot k^3$$

With CRT

$$t_{DEC\text{-}CRT}(k) \approx 2 \cdot t_{EXP}(\text{random } e, k/2, L{=}k/2) = 2 \cdot c_s \cdot \left(\frac{k}{2}\right)^3 = \underline{\underline{\frac{1}{4}\, t_{DEC}(k)}}$$

HARDWARE

Without CRT

$$t_{DEC}(k) = t_{EXP}(\text{random } e, k, L{=}k) = c_h \cdot k^2$$

With CRT

$$t_{DEC\text{-}CRT}(k) \approx t_{EXP}(\text{random } e, k/2, L{=}k/2) = c_h \cdot \left(\frac{k}{2}\right)^2 = \underline{\underline{\frac{1}{4}\, t_{DEC}(k)}}$$

Chinese Remainder Theorem

Let

$$\mathbf{N = n_1 \cdot n_2 \cdot n_3 \; \ldots \cdot n_M}$$

and

for any i, j $\quad \gcd(n_i, n_j) = 1$

Then, any number $\quad 0 \le A \le N-1$

can be represented uniquely by

$$\mathbf{A \leftrightarrow (a_1 = A \bmod n_1,\; a_2 = A \bmod n_2,\; \ldots,\; a_M = A \bmod n_M)}$$

A can be reconstructed from $(a_1, a_2, \ldots, a_M)$ using equation

$$\mathbf{A = \sum_{i=1}^{M} (a_i \cdot N_i \cdot N_i^{-1} \bmod n_i) \bmod N}$$

where $N_i = \dfrac{N}{n_i} = n_1 \cdot n_2 \cdot \ldots \cdot n_{i-1} \cdot n_{i+1} \cdot \ldots \cdot n_M$

Chinese Remainder Theorem for N=P · Q

$N = P \cdot Q$ $\qquad$ $\gcd(P, Q) = 1$

$$\mathbf{M \leftrightarrow (M_p = M \bmod P,\ M_Q = M \bmod Q)}$$

$$M = M_P \cdot \frac{N}{P} \cdot \left[\left[\frac{N}{P}\right]^{-1} \bmod P\right] + M_Q \cdot \frac{N}{Q} \cdot \left[\left[\frac{N}{Q}\right]^{-1} \bmod Q\right] \bmod N$$

$$= M_P \cdot Q \cdot ((Q^{-1}) \bmod P) + M_Q \cdot P \cdot ((P^{-1}) \bmod Q) \bmod N =$$

$$= M_P \cdot R_Q + M_Q \cdot R_P \bmod N$$

Concealment of messages in the RSA cryptosystem

Blakley, Borosh, 1979

There exist messages that are not changed by the RSA encryption!

For example:

M=1 $\qquad$ $C = 1^e \bmod N = 1$
M=0 $\qquad$ $C = 0^e \bmod N = 0$
M=n-1≡-1 mod N $\qquad$ $C = (-1)^e \bmod N = -1$

Every M such that

$M_p = M \bmod p \in \{1, 0, -1\}$
$M_q = M \bmod q \in \{1, 0, -1\}$

$C_p = C \bmod p = M^e \bmod p = M_p^e \bmod p = M_p$
$C_q = C \bmod q = M^e \bmod q = M_q^e \bmod q = M_q$

Concealment of messages in the RSA cryptosystem

Blakley, Borosh, 1979

At least 9 messages not concealed by RSA!

Number of messages not concealed by RSA:

$$\sigma = (1 + \gcd(e\text{-}1, p\text{-}1)) \cdot (1 + \gcd(e\text{-}1, q\text{-}1))$$

A. $e=3$ $\qquad \sigma = 9$

B. $\gcd(e\text{-}1, p\text{-}1) = 2$ and $\gcd(e\text{-}1, q\text{-}1) = 2$ $\qquad \sigma = 9$

C. $\gcd(e\text{-}1, p\text{-}1) = p\text{-}1$ and $\gcd(e\text{-}1, q\text{-}1) = q\text{-}1$ $\qquad \sigma = p \cdot q = N$

It is possible that all messages remain unconcealed by RSA!

Generation of the RSA keys

e

Typically
$e = 3$ or
$e = 2^{16} + 1$

prime number generation

P, Q

gcd(e, P-1) = 1
gcd(e, Q-1) = 1

gcd(e-1, P-1) = 2
gcd(e-1, Q-1) = 2

Extended Euclid's algorithm

$N = P \cdot Q$

$d = e^{-1} \bmod (P\text{-}1) \cdot (Q\text{-}1)$

RSA – countermeasures against known attacks

Wiener's attack

If $d < N^{1/4}$

N

d

d can be mathematically reconstructed from e and N

Countermeasure:

Choose e, p, and q first
Compute $d = e^{-1} \bmod (p-1)(q-1)$
Check if $d > N^{1/4}$

Recovering RSA-encrypted messages without a private key (1)

Guessing a set of possible messages

$$\text{IRS} \longrightarrow \text{FBI}$$

$E_{\text{public\_key\_of\_FBI}}$(name of the congress member who committed a tax fraud)

---

journalist

$E_{\text{public\_key\_of\_FBI}}$ (name1)
$E_{\text{public\_key\_of\_FBI}}$ (name2)
………………………..
$E_{\text{public\_key\_of\_FBI}}$ (nameN)

Recovering RSA-encrypted messages without a private key (2)

Small e and small messages

e=3

00000000000000000	m

$$m < N^{1/3}$$

$$c = m^3 \bmod N = m^3 \xrightarrow{1/3} m$$

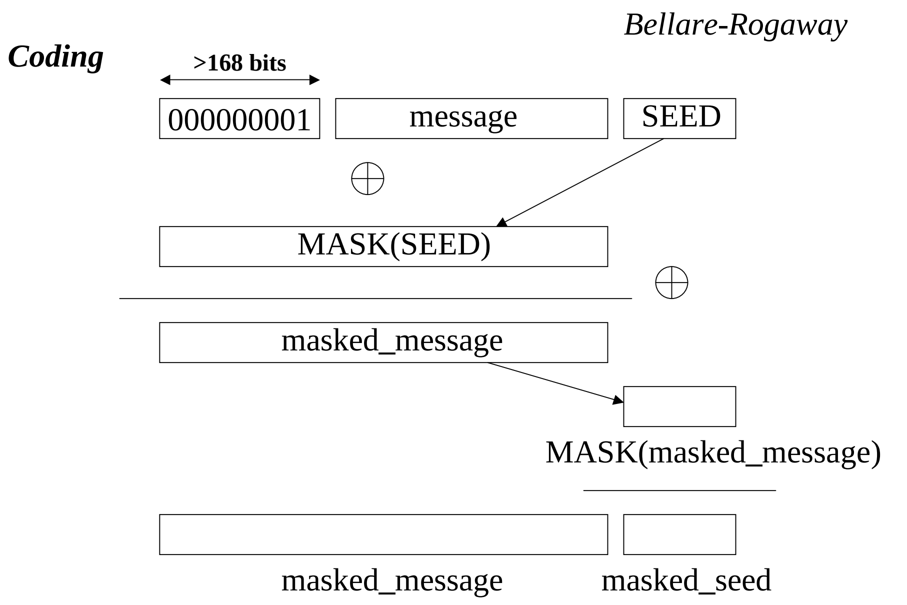
Hastad's attack

e=3, m send to three different people

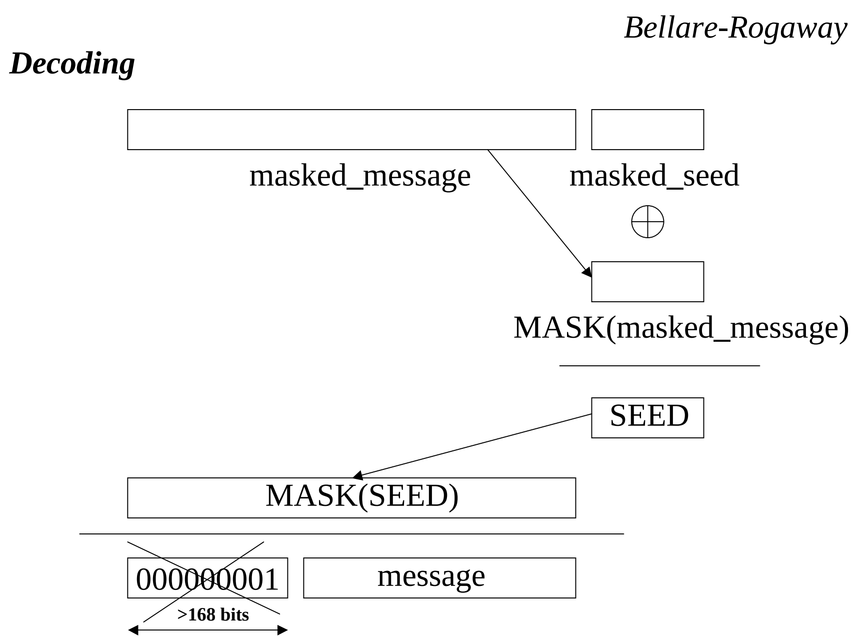
$P_{U1} = (3, N_1)$ $m^3 \bmod N_1$
$P_{U2} = (3, N_2)$ $m^3 \bmod N_2$
$P_{U3} = (3, N_3)$ $m^3 \bmod N_3$

$$\xrightarrow{\text{CRT}} m^3 \bmod N_1N_2N_3 = m^3 \xrightarrow{1/3} m$$

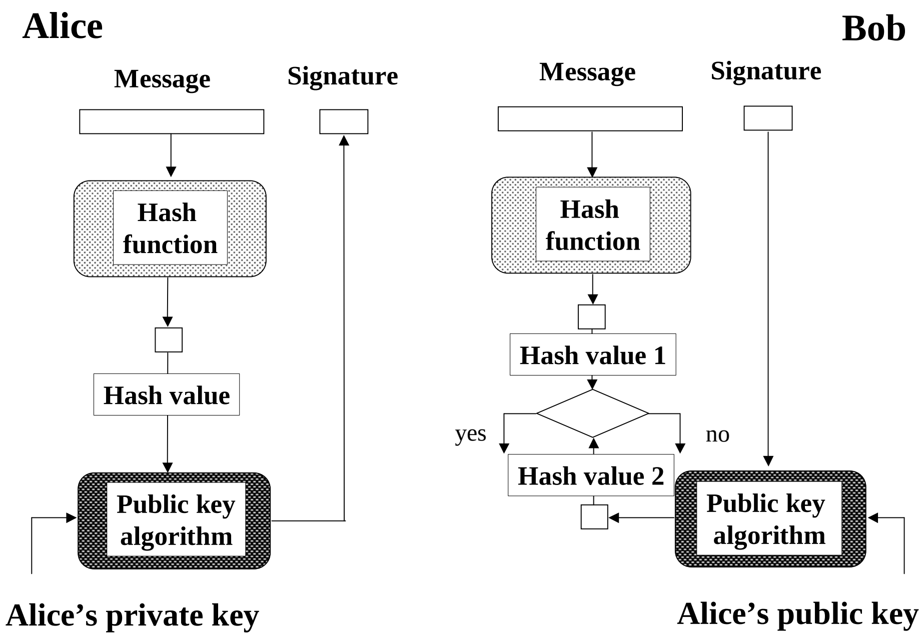
Optimal Assymetric Encryption Padding (1)



Optimal Assymetric Encryption Padding (2)



RSA signature



Padding for signatures with appendix

PKCS #1 for signatures

00 01	FF FF FF …. FF	00	h(m)

at least 8 bytes

ISO-14888

6	BBBBBBBBBBB	A	h(m)	**hash code**

33CC for SHA-1
31CC for RIPEMD-160

Superencryption attack

Simmons, Norris, 1977

$C_0 = C$
$C_1 = C_0^e \bmod N$
$C_2 = C_1^e \bmod N$
……………………

$\mathbf{C_{k-1}} = C_{k-2}^e \bmod N$
$C_k = C_{k-1}^e \bmod N = C_0 = C$

$\mathrm{M} = C_{k-1}$ because $M^e \bmod N = C$

Superencryption attack

Simmons, Norris, 1977

Typically, number of iterations very large if p and q chosen at random

Additional protection may be achieved if:

p-1 has a large prime factor r_p
q-1 has a large prime factor r_q
r_p-1 has a large prime factor t_p
r_q-1 has a large prime factor t_q

$e^{(rp-1)/tp} \bmod r_p \neq 1$
$e^{(rq-1)/tq} \bmod r_q \neq 1$

For these conditions

# of iterations, $k \geq t_p \cdot t_q$

Strong primes

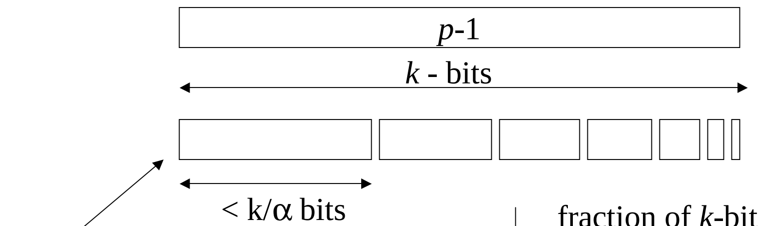
Gordon algorithm, based on CRT,
allows to generate strong primes

time to generate a *strong* prime = 1.2 · time to generate a *regular* prime

Only 20 % increase in time

Strong primes

Most of the large primes generated <u>at random</u> are strong anyway!



Largest prime factor of *p*-1

$$\alpha = \frac{\text{\# bits of } n}{\text{\# bits of the largest prime factor}}$$

α	fraction of *k*-bit numbers whose *largest* prime factor has *less* than k/α bits
2	31%
3	5%
4	0.5%
5	0.035%
9	0.0000001%

Factoring methods

General purpose	Special purpose
Time of factoring depends only on the size of N	*Time of factoring is much shorter if N or factors of N are of the special form*
GNFS - General Number Field Sieve	ECM - Elliptic Curve Method
QS - Quadratic Sieve	Pollard's p-1 method
Continued Fraction Method *(historical)*	Cyclotomic polynomial method
	SNFS - Special Number Field Sieve

Special purpose factoring methods

Name	Condition for a speed-up
ECM - Elliptic Curve Method	One of the factors of N is smaller than 40-45 decimal digits
Pollard's p-1 method	N has a prime factor p such that p-1 is B-smooth with respect to some relatively small bound B p-1 is B-smooth if $p-1 = p_1^{e1} p_2^{e2} \cdot \ldots \cdot p_k^{ek}$, where $p_i < B$ for all i
Cyclotomic polynomial method	N has a prime factor p such that p+1 is B-smooth with respect to some relatively small bound B
Special Number Field Sieve - SNFS	N is of the form $r^e - s$ for small r and $\lvert s \rvert$

RSA for paranoids
Rationale

Shamir 1995

Size of N (k)

500 bits → 5000 bits
150 D → 1500 D

Time of decipherment $t_{DEC} = c \cdot k^3$

k	increases	10 times	(500→5000)
t_{DEC}	increases	1000 times	(1 s → 16 min)

RSA for paranoids
Solution (1)

Shamir 1995

Choose

p - 500 bits q - 4500 bits

N - 5000 bits (k=5000)

Security:

As resistant as classical RSA with k=5000 against general purpose factoring.

Sufficiently resistant against known special purpose methods.

Decryption using Chinese Remainder Theorem

$$M = C^{d} \bmod N$$

$C_P = C \bmod P$	$C_Q = C \bmod Q$
$d_P = d \bmod (P-1)$	$d_Q = d \bmod (Q-1)$
$M_P = C_P^{\,d_P} \bmod P$	$M_Q = C_Q^{\,d_Q} \bmod Q$

$$M = M_P \cdot R_Q + M_Q \cdot R_P \bmod N$$

where

$$R_P = (P^{-1} \bmod Q) \cdot P = P^{Q-1} \bmod N$$

$$R_Q = (Q^{-1} \bmod P) \cdot Q = Q^{P-1} \bmod N$$

RSA for paranoids
Solution (2)

Shamir 1995

Make

$M \in (0, p\text{-}1)$ 500 bits
$e \in (20, 100)$ 5-7 bits
$d \in (0, \varphi(N))$ 5000 bits

Ciphering:

$C = M^{e} \bmod N$

Deciphering:

$M_p = C_p^{\,dp} \bmod p = M \bmod p = M$

where $C_p = C \bmod p$
$d_p = d \bmod p\text{-}1$

Efficiency:

Time of deciphering the same as in regular RSA with k=500