

# Cryptography

## Course 9: 30 years of attacks against RSA

Jean-Sébastien Coron

Université du Luxembourg

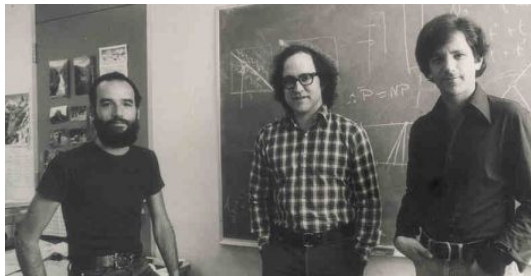
May 5, 2011

# Public-key encryption

- Public-key encryption: two keys.
  - One key is made public and used to encrypt.
  - The other key is kept private and enables to decrypt.
- Alice wants to send a message to Bob:
  - She encrypts it using Bob's public-key.
  - Only Bob can decrypt it using his own private-key.
  - Alice and Bob do not need to meet to establish a secure communication.
- Security:
  - It must be difficult to recover the private-key from the public-key
  - but not enough in practice.

# The RSA algorithm

- The RSA algorithm is the most widely-used public-key encryption algorithm
  - Invented in 1977 by Rivest, Shamir and Adleman.
  - Used for encryption and signature.
  - Widely used in electronic commerce protocols (SSL).



- Key generation:
  - Generate two large distinct primes  $p$  and  $q$  of same bit-size.
  - Compute  $n = p \cdot q$  and  $\phi = (p - 1)(q - 1)$ .
  - Select a random integer  $e$ ,  $1 < e < \phi$  such that  $\gcd(e, \phi) = 1$
  - Compute the unique integer  $d$  such that

$$e \cdot d \equiv 1 \pmod{\phi}$$

using the extended Euclidean algorithm.

- The public key is  $(n, e)$ . The private key is  $d$ .

- Encryption

- Given a message  $m \in [0, n - 1]$  and the recipient's public-key  $(n, e)$ , compute the ciphertext:

$$c = m^e \pmod n$$

- Decryption

- Given a ciphertext  $c$ , to recover  $m$ , compute:

$$m = c^d \pmod n$$

- Definition:
  - $\phi(n)$  for  $n > 0$  is defined as the number of integers  $a$  comprised between 0 and  $n - 1$  such that  $\gcd(a, n) = 1$ .
  - $\phi(1) = 1, \phi(2) = 1, \phi(3) = 2, \phi(4) = 2$ .
- Equivalently:
  - Let  $\mathbb{Z}_n^*$  be the set of integers  $a$  comprised between 0 and  $n - 1$  such that  $\gcd(a, n) = 1$ .
  - Then  $\phi(n) = |\mathbb{Z}_n^*|$ .

- If  $p \geq 2$  is prime, then

$$\phi(p) = p - 1$$

- More generally, for any  $e \geq 1$ ,

$$\phi(p^e) = p^{e-1} \cdot (p - 1)$$

- For  $n, m > 0$  such that  $\gcd(n, m) = 1$ , we have:

$$\phi(n \cdot m) = \phi(n) \cdot \phi(m)$$

- Theorem

- For any integer  $n > 1$  and any integer  $a$  such that  $\gcd(a, n) = 1$ , we have  $a^{\phi(n)} \equiv 1 \pmod n$ .

- Proof

- Consider the map  $f : \mathbb{Z}_n^* \rightarrow \mathbb{Z}_n^*$ , such that  $f(b) = a \cdot b$  for any  $b \in \mathbb{Z}_n^*$ .
- $f$  is a permutation, therefore :

$$\prod_{b \in \mathbb{Z}_n^*} b = \prod_{b \in \mathbb{Z}_n^*} (a \cdot b) = a^{\phi(n)} \cdot \left( \prod_{b \in \mathbb{Z}_n^*} b \right)$$

- Therefore, we obtain  $a^{\phi(n)} \equiv 1 \pmod n$ .

# Fermat's little theorem

- Theorem

- For any prime  $p$  and any integer  $a \not\equiv 0 \pmod{p}$ , we have  $a^{p-1} \equiv 1 \pmod{p}$ . Moreover, for any integer  $a$ , we have  $a^p \equiv a \pmod{p}$ .

- Proof

- Follows from Euler's theorem and  $\phi(p) = p - 1$ .

# Proof that decryption works

- Since  $e \cdot d \equiv 1 \pmod{\phi}$ , there is an integer  $k$  such that  $e \cdot d = 1 + k \cdot \phi$ .
- If  $m \not\equiv 0 \pmod{p}$ , then by Fermat's little theorem  $m^{p-1} \equiv 1 \pmod{p}$ , which gives :

$$m^{1+k \cdot (p-1) \cdot (q-1)} \equiv m \pmod{p}$$

- This equality is also true if  $m \equiv 0 \pmod{p}$ .
- This gives  $m^{ed} \equiv m \pmod{p}$  for all  $m$ .
- Similarly,  $m^{ed} \equiv m \pmod{q}$  for all  $m$ .
- By the Chinese Remainder Theorem, if  $p \neq q$ , then

$$m^{ed} \equiv m \pmod{n}$$

# The RSA signature scheme

- Key generation :
  - Public modulus:  $N = p \cdot q$  where  $p$  and  $q$  are large primes.
  - Public exponent :  $e$
  - Private exponent:  $d$ , such that  $d \cdot e = 1 \pmod{\phi(N)}$
- To sign a message  $m$ , the signer computes :
  - $s = m^d \pmod{N}$
  - Only the signer can sign the message.
- To verify the signature, one checks that:
  - $m = s^e \pmod{N}$
  - Anybody can verify the signature

- There are many attacks on basic RSA signatures:
  - Existential forgery:  $r^e = m \pmod N$
  - Chosen-message attack:  $(m_1 \cdot m_2)^d = m_1^d \cdot m_2^d \pmod N$
- To prevent from these attacks, one usually uses a hash function. The message is first hashed, then padded.
  - $m \rightarrow H(m) \rightarrow 1001\dots0101\|H(m)$
  - Example: PKCS#1 v1.5:  
 $\mu(m) = 0001\text{ FF}\dots\text{FF}00\|c_{\text{SHA}}\|\text{SHA}(m)$
  - ISO 9796-2:  $\mu(m) = 6A\|m[1]\|H(m)\|BC$
  - The signature is then  $\sigma = \mu(m)^d \pmod N$

# Attacks against RSA

- Factoring
  - Equivalence between factoring and breaking RSA ?
- Mathematical attacks
  - Attacks against plain RSA encryption and signature
  - Heuristic countermeasures
  - Low private / public exponent attacks
  - Provably secure constructions
- Implementation attacks
  - Timing attacks, power attacks and fault attacks
  - Countermeasures

- Factoring large integers
  - Best factoring algorithm: Number Field Sieve
  - Sub-exponential complexity

$$\exp\left((c + o(1)) n^{1/3} \log^{2/3} n\right)$$

for  $n$ -bit integer.

- Current factoring record: 768-bit RSA modulus.
- Use at least 1024-bit RSA moduli
  - 2048-bit for long-term security.

# Factoring vs breaking RSA

- Breaking RSA:
  - Given  $(N, e)$  and  $y$ , find  $x$  such that  $y = x^e \pmod N$
- Open problem
  - Is breaking RSA equivalent to factoring ?
- Knowing  $d$  is equivalent to factoring
  - Probabilistic algorithm (RSA, 1978)
  - Deterministic algorithm (A. May 2004, J.S. Coron and A. May 2007)

- Plain RSA encryption: dictionary attack
  - If only two possible messages  $m_0$  and  $m_1$ , then only  $c_0 = (m_0)^e \bmod N$  and  $c_1 = (m_1)^e \bmod N$ .
  - $\Rightarrow$  encryption must be probabilistic.
- PKCS#1 v1.5
  - $\mu(m) = 0002\|r\|00\|m$
  - $c = \mu(m)^e \bmod N$
  - Still insufficient (Bleichenbacher's attack, 1998)

# Attacks against Plain RSA signature

- Existential forgery
  - $r^e = m \pmod N$ , so  $r$  is signature of  $m$
- Chosen message attack
  - $(m_1 \cdot m_2)^d = m_1^d \cdot m_2^d \pmod N$
- To prevent from these attacks, one first computes  $\mu(m)$ , and lets  $s = \mu(m)^d \pmod N$ 
  - ISO 9796-1:

$$\mu(m) = \bar{s}(m_z)s(m_{z-1})m_z m_{z-1} \dots s(m_1)s(m_0)m_0 6$$

- ISO 9796-2:

$$\mu(m) = 6A\|m[1]\|H(m)\|BC$$

- PKCS#1 v1.5:

$$\mu(m) = 0001\text{ FF} \dots \text{FF}00\|c_{\text{SHA}}\|\text{SHA}(m)$$

# Attacks against RSA signatures

- Desmedt and Odlyzko attack (Crypto 85)
  - Based on finding messages  $m$  such that  $\mu(m)$  is smooth (product of small primes only)
  - $\mu(m_i) = \prod_j p_j^{\alpha_{i,j}}$  for many messages  $m_i$ .
  - Solve a linear system and write  $\mu(m_k) = \prod_i \mu(m_i)$
  - Then  $\mu(m_k)^d = \prod_i \mu(m_i)^d \pmod N$
- Application to ISO 9796-1 and ISO 9796-2 signatures
  - Cryptanalysis of ISO 9796-1 (Coron, Naccache, Stern, 1999)
  - Cryptanalysis of ISO 9796-2 (Coron, Naccache, Tibouchi, Weinmann, 2009)
  - Extension of Desmedt and Odlyzko attack.
  - For ISO 9796-2 the attack is feasible if the output size of the hash function is small enough.

# Low private exponent attacks

- To reduce decryption time, one could use a small  $d$ 
  - Wiener's attack: recover  $d$  if  $d < N^{0.25}$
- Boneh and Durfee's attack (1999)
  - Recover  $d$  if  $d < N^{0.29}$
  - Based on lattice reduction and Coppersmith's technique
  - Open problem: extend to  $d < N^{0.5}$
- Conclusion: devastating attack
  - Use a full-size  $d$

# Low public exponent attack

- To reduce encryption time, one can use a small  $e$ 
  - For example  $e = 3$  or  $e = 2^{16} + 1$
- Coppersmith's theorem :
  - Let  $N$  be an integer and  $f$  be a polynomial of degree  $\delta$ .  
Given  $N$  and  $f$ , one can recover in polynomial time all  $x_0$  such that  $f(x_0) = 0 \pmod N$  and  $x_0 < N^{1/\delta}$ .
- Application: partially known message attack :
  - If  $c = (B||m)^3 \pmod N$ , one can recover  $m$  if  $|m| < |N|/3$
  - Define  $f(x) = (B \cdot 2^k + x)^3 - c \pmod N$ .
  - Then  $f(m) = 0 \pmod N$  and apply Coppersmith's theorem to recover  $m$ .

- Coppersmith's short pad attack
  - Let  $c_1 = (m||r_1)^3 \pmod N$  and  $c_2 = (m||r_2)^3 \pmod N$
  - One can recover  $m$  if  $r_1, r_2 < N^{1/9}$
  - Let  $g_1(x, y) = x^3 - c_1$  and  $g_2(x, y) = (x + y)^3 - c_2$ .
  - $g_1$  and  $g_2$  have a common root  $(m||r_1, r_2 - r_1)$  modulo  $N$ .
  - $h(y) = \text{Res}_x(g_1, g_2)$  has a root  $\Delta = r_2 - r_1$ , with  $\deg h = 9$ .
  - To recover  $m||r_1$ , take gcd of  $g_1(x, \Delta)$  and  $g_2(x, \Delta)$ .
- Conclusion:
  - Attack only works for particular encryption schemes.
  - Low public exponent is secure when provably secure construction is used. One often takes  $e = 2^{16} + 1$ .

# Implementation attacks

- The implementation of a cryptographic algorithm can reveal more information
- Passive attacks :
  - Timing attacks (Kocher, 1996): measure the execution time
  - Power attacks (Kocher et al., 1999): measure the power consumption
- Active attacks :
  - Fault attacks (Boneh et al., 1997): induce a fault during computation
  - Invasive attacks: probing.

- Described on RSA by Kocher at Crypto 96.
  - Let  $d = \sum_{i=0}^n 2^i d_i$ .
  - Computing  $m^d \bmod N$  using square and multiply :
    - Let  $z \leftarrow m$   
For  $i = n - 1$  downto 0 do  
  Let  $z \leftarrow z^2 \bmod N$   
  If  $d_i = 1$  let  $z \leftarrow z \cdot m \bmod N$
- Attack
  - Let  $T_i$  be the total time needed to compute  $m_i^{d_i} \bmod N$
  - Let  $t_i$  be the time needed to compute  $m_i^3 \bmod N$
  - If  $d_{n-1} = 1$ , the variables  $t_i$  and  $T_i$  are correlated, otherwise they are independent. This gives  $d_{n-1}$ .

- Implement in constant time
  - Not always possible with hardware crypto-processors.
- Exponent blinding:
  - Compute  $m^{d+k\cdot\phi(N)} = m^d \pmod N$  for random  $k$ .
- Message blinding
  - Compute  $(m \cdot r)^d / r^d = m^d \pmod N$  for random  $r$ .
- Modulus randomization
  - Compute  $m^d \pmod{(N \cdot r)}$  and reduce modulo  $N$ .
- or a combination of the three.

- Based on measuring power consumption
  - Introduced by Kocher *et al.* at Crypto 99.
  - Initially applied on DES, but any cryptographic algorithm is vulnerable.
- Attack against exponentiation  $m^d \bmod N$  :
  - If power consumption correlated with some bits of  $m^3 \bmod N$ , this means that  $m^3 \bmod N$  was effectively computed, and so  $d_{n-1} = 1$ .
  - Enables to recover  $d_{n-1}$  and by recursion the full  $d$ .

- Hardware countermeasures
  - Constant power consumption; dual rail logic.
  - Random delays to desynchronise signals.
- Software countermeasures
  - Same as for timing attacks
  - Goal: randomization of execution
  - Drawback: increases execution time.

- Induce a fault during computation
  - By modifying voltage input
- RSA with CRT: to compute  $s = m^d \pmod N$ , compute :
  - $s_p = m^{d_p} \pmod p$  where  $d_p = d \pmod{p-1}$
  - $s_q = m^{d_q} \pmod q$  where  $d_q = d \pmod{q-1}$
  - and recombine  $s_p$  and  $s_q$  using CRT to get  $s = m^d \pmod N$
- Fault attack against RSA with CRT (Boneh *et al.*, 1996)
  - If  $s_p$  is incorrect, then  $s^e \neq m \pmod N$  while  $s^e = m \pmod q$
  - Therefore,  $\gcd(N, s^e - m)$  gives the prime factor  $q$ .