How to implement RSA in practice

Part 2

Jean-Sébastien Coron

Université du Luxembourg

October 12, 2009

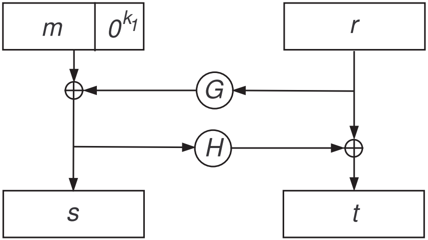
How to implement RSA in practice

- The RSA algorithm (previous course)
  - Key generation, encryption, decryption
  - Mathematical attacks against RSA
- Provably secure constructions
  - Encryption
  - Signature
- Implementation attacks
  - Timing attacks
  - Power attacks
  - Fault attacks

Provable security for RSA encryption

- Security notion for encryption.
  - From a ciphertext c, an attacker should not be able to derive any information from the corresponding plaintext m.
  - Even if the attacker can obtain the decryption of any ciphertext, c excepted.
  - This is called indistinguishability against a chosen ciphertext attack (IND-CCA2).
- Security proof for encryption
  - Prove that if an attacker can distinguish between the encryption of two plaintexts, then it can be used to break RSA.

- OAEP (Bellare and Rogaway, E'94)
  - IND-CCA2, assuming that RSA is hard to invert.
  - PKCS #1 v2.1



$$c = (s\|t)^e \mod N$$

Provable security for signature

- Strongest security notion (Goldwasser, Micali and Rivest, 1988):
  - It must be infeasible for an adversary to forge the signature of a message, even if he can obtain the signature of messages of his choice.
- Security proof:
  - Show that from an adversary who is able to forge signature, one can solve a difficult problem, such as inverting RSA.
- Examples of provably secure signature schemes:
  - Full Domain Hash (FDH)
  - Probabilistic Signature Scheme (PSS)

The FDH scheme

- The FDH signature scheme:
  - was designed in 1993 by Bellare and Rogaway.

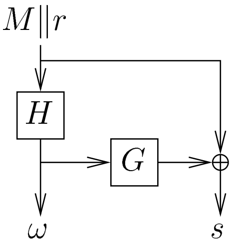
$$m \longrightarrow H(m) \longrightarrow s = H(m)^d \mod N$$

  - The hash function $H(m)$ has the same output size as the modulus.
- Security of FDH
  - FDH is provably secure in the random oracle model, assuming that inverting RSA is hard.
  - In the random oracle model, the hash function is replaced by an oracle which outputs a random value for each new query.

The PSS signature cheme

- PSS (Bellare and Rogaway, Eurocrypt'96)
  - IEEE P1363a and PKCS#1 v2.1.
  - 2 variants: PSS and PSS-R (message recovery)
  - Provably secure against chosen-message attacks
  - PSS-R:

$$\sigma = \mu(M, r)^d \mod N = (\omega \| s)^d \mod N$$



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.

Timing attacks

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

Countermeasures

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

Power attacks

- 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 $m^d \mod N$:
  - If power consumption correlated with some bits of $m^3 \mod N$, this means that $m^3 \mod N$ was effectively computed, and so $d_{n-1} = 1$.
  - Enables to recover d_{n-1} and by recursion the full d.

Countermeasures

- 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.

Fault attacks

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

Conclusion

- Thirty years of attacks against RSA
  - No devastating attack against RSA, but illustrate numerous pitfalls.
- Mathematical attacks
  - Use provably secure constructions
  - with a large enough modulus.
- Implementation attacks:
  - Designing countermeasures requires expertise in electronics, signal analysis, hardware design and cryptography.
  - In practice, compromise between security, efficiency and patents.