

Foundations of Computer Security

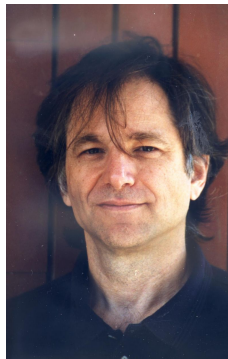
Andrea Pintér-Husztí

Contents

- RSA encryption scheme
- Security Analysis

RSA encryption scheme

RSA encryption scheme



- Appeared in 1977
- Designers: Ron Rivest, Adi Shamir, and Leonard Adleman
- Most Public Key Infrastructure (PKI) products
- SSL/TLS: Certificates and key-exchange
- Secure e-mail: PGP, Outlook

RSA encryption scheme

Asymmetric encryption scheme: $AE = (Key, Enc, Dec)$

- *Key*:

- 1 Randomly choose two large primes: p, q .
- 2 Calculate RSA modulus: $n = p \cdot q$.
- 3 Calculate Euler totient: $\phi(n) = (p - 1)(q - 1)$.
- 4 Randomly choose an integer e : $1 < e < \phi(n)$ and $(e, \phi(n)) = 1$. (e is the encryption exponent)
- 5 Calculate d : $1 < d < \phi(n)$ and $ed \equiv 1 \pmod{\phi(n)}$. (d is the decryption exponent)

$PK = (n, e)$, $SK = d$ and $\phi(n), p, q$ are secret parameters

$$\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$$

- $Enc_{PK}(m) = m^e \pmod{n} \quad \forall m \in \mathcal{P}$ and for a $PK = (n, e)$.
- $Dec_{SK}(c) = c^d \pmod{n} \quad \forall c \in \mathcal{C}$ and for a $SK = d$.

An example

Asymmetric encryption scheme: $AE = (Key, Enc, Dec)$

- *Key*:
 - 1 Randomly choose two large primes: $p = 5, q = 11$.
 - 2 Calculate RSA modulus: $n = p \cdot q = 55$.
 - 3 Calculate Euler totient: $\phi(n) = (p - 1)(q - 1) = 40$.
 - 4 Randomly choose an integer e : $1 < e < \phi(n)$ and $(e, \phi(n)) = 1$, $e = 3$.
 - 5 Calculate d : $1 < d < \phi(n)$ and $ed \equiv 1 \pmod{\phi(n)}$, $d = 27$.

$PK = (n = 55, e = 3)$, $SK = d = 27$ and

$\phi(n) = 40, p = 5, q = 11$ are secret parameters

$\mathcal{P} = \mathcal{C} = \mathbb{Z}_{55}$

- $m = 8$ and $PK = (55, 3)$: $Enc_{PK}(m) = 8^3 \pmod{55}$
 $8^3 \equiv 17 \pmod{55}$
- $c = 17$ and $SK = 27$: $Dec_{SK}(c) = 17^{27} \pmod{55}$
 $17^{27} \equiv 8 \pmod{55}$

- *Key:*
 - 1 Randomly choose two large primes: p, q . -> **Primality tests** (e.g. Miller-Rabin primality test)
 - 2 Randomly choose an integer e : $1 < e < \phi(n)$ and $(e, \phi(n)) = 1$. -> **Euclidean algorithm**
 - 3 Calculate d : $1 < d < \phi(n)$ and $ed \equiv 1 \pmod{\phi(n)}$. -> calculating multiplicative inverse: **Extended Euclidean algorithm**
- $Enc_{PK}(m) = m^e \pmod{n} \forall m \in \mathcal{P}$ and for a $PK = (n, e)$. -> **Fast exponentiation**
- $Dec_{SK}(c) = c^d \pmod{n} \forall c \in \mathcal{C}$ and for a $SK = d$. -> **Chinese Remainder theorem**

Security Analysis

Infeasible to calculate SK from PK $\longrightarrow$ prime factorization

Adversary's goal is to calculate the secret key:

- Theorem: Computing d from (n, e) is **as hard as** finding the prime factors p and q of n , where $|p| = |q|$.

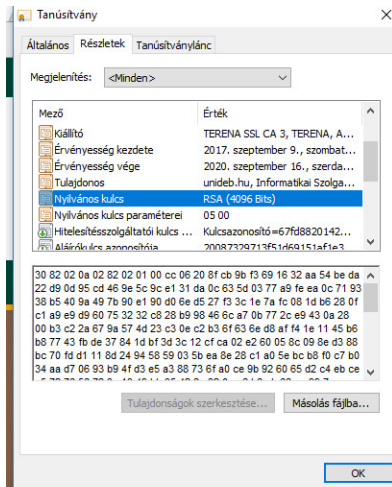
Comments:

- If we can calculate the factors of n efficiently, then there is an efficient algorithm for calculating d from $ed \equiv 1 \pmod{\phi(n)}$. To compute $\phi(n)$, primes p and q are necessary.
- If there is an efficient algorithm for calculating d from (n, e) , we can give an efficient algorithm for factoring n .

Prime factorization

- There is no proof that factoring RSA modules is difficult. However, if the factors p and q of the RSA module n are sufficiently large, then nobody yet knows how to factor n .
- Size of the parameters (bits): $|n| = 768$, with $|p| = |q| = 384$ is factorized in 2009 with Number Field Sieve technique.
 $|n| = 1024$, with $|p| = |q| = 512$ is still secure.
<https://unideb.hu> $\longrightarrow$ $|n| = 4096$ (archive for future)

Size of the RSA modulus (2018)



Infeasible to calculate m from c $\longrightarrow$ RSA problem

Adversary's goal is to calculate the plaintext:

- **The RSA Problem:** Given an RSA public key (n, e) and a ciphertext $c \equiv m^e \pmod{n}$, to compute m .
- The adversary may not know the secret key, hence the adversary should calculate: $c^{\frac{1}{e}} \pmod{n}$
- The **RSA Assumption** is that the RSA Problem is hard to solve when the modulus n is sufficiently large and randomly generated, and the plaintext m (and hence the ciphertext c) is a random integer between 0 and $n - 1$.
- The randomness of the plaintext m over the range $[0, n - 1]$ is important in the assumption. If m is known to be from a small space, for instance, then an adversary can solve for m by trying all possible values for m .

Prime factorization vs. RSA problem

- Prime factorization $\rightarrow$ RSA problem

The RSA Problem is clearly no harder than integer factoring, since an adversary who can factor the modulus n can compute the private key (n, d) from the public key (n, e) .

- RSA problem $\rightarrow$ Prime factorization

However, it is not clear whether the converse is true, that is, whether an algorithm for integer factoring can be efficiently constructed from an algorithm for solving the RSA Problem.

Attacks against textbook RSA

- Special Message Spaces

Attack: Ciphertext Only Attack (COA)

Input: $c, (n, e)$

Output: m , where $m^e \equiv c \pmod{n}$

Algorithm: Try each possible plaintext from the special message space. (Brute Force)

- Relationship between Encrypted Messages

Attack: Chosen Ciphertext Attack (CCA)

Input: $c, (n, e)$ and m' for a chosen c'

Output: m

Algorithm:

- 1 Randomly choose $r \in \mathcal{P}$
- 2 Calculate $r' \equiv r^e \pmod{n}$, ask $c' \equiv r' \cdot c \pmod{n}$ to decrypt
- 3 m' is received, where $m' \equiv c'^d \equiv (r' \cdot c)^d \equiv r \cdot m \pmod{n}$
- 4 $m \equiv m' \cdot r^{-1} \pmod{n}$

RSA-OAEP

- The textbook RSA is a completely insecure cryptosystem. Does not satisfy basic definitions of security. Never use textbook RSA.
- In practise: RSA-OAEP (Optimal Asymmetric Encryption Padding):

Given: $G : \{0, 1\}^k \rightarrow \{0, 1\}^l$, $H : \{0, 1\}^l \rightarrow \{0, 1\}^k$,

$\mathcal{P} = \{0, 1\}^l$

Encryption:

- Input: $m \in \{0, 1\}^l$ and $r \in \{0, 1\}^k$
- Output: $c = ((m \oplus G(r)) || (r \oplus H(m \oplus G(r))))^e \pmod n$

Decryption:

- Input: $c \in \{0, 1\}^{l+k}$
- Calculate:
 - $c^d = ((m \oplus G(r)) || (r \oplus H(m \oplus G(r)))) \pmod n$
 - Determine r : $r = (r \oplus H(m \oplus G(r))) \oplus H(m \oplus G(r))$
 - $m = (m \oplus G(r)) \oplus G(r)$
- Output: m

One-way function

- One-way function:
 - **Easy to compute:** Given x , it is easy to compute $f(x)$.
 - **Hard to invert:** Given $f(x)$, it is infeasible to calculate x .
- There is no proof that one-way functions exist, or even real evidence that they can be constructed.
- One-way function candidate: **multiplication of two primes p, q , with $|p| = |q|$**
 - **Easy to compute:** Given p, q , it is easy to compute $f(p, q) = p \cdot q$.
 - **Hard to invert:** Given $f(n) = p \cdot q$, it is infeasible to calculate p or/and q . (prime factorization)

Trapdoor one-way function

- Trapdoor one-way function:
 - One-way function
 - **Easy to compute:** Given x , it is easy to compute $f(x)$.
 - **Hard to invert:** Given $f(x)$, it is infeasible to calculate x .
 - Trapdoor information: With certain knowledge it is easy to invert, to calculate x from $f(x)$.
- There is no proof that trapdoor one-way functions exist, or even real evidence that they can be constructed.
- Trapdoor one-way function candidate: **modular exponentiation**
 - One-way function
 - **Easy to compute:** Given m , it is easy to compute $f(m) = m^e \pmod{n}$ (fast exponentiation algorithm).
 - **Hard to invert:** Given $f(m) = m^e \pmod{n}$, it is infeasible to calculate m . (RSA Assumption)
 - Trapdoor information: $d, p, q, \phi(n)$.