Cryptography

Public-Key Encryption

Prof. Dr. Heiko Knospe

TH Köln – University of Applied Sciences

July 17, 2019

Encryption scheme

Definition

A *public-key encryption scheme* (public-key cryptosystem) is given by:

- A plaintext space $\mathcal{M}$ and a ciphertext space $\mathcal{C}$,
- A key space $\mathcal{K} = \mathcal{K}_{pk} \times \mathcal{K}_{sk}$ (pairs of public and private keys),
- A randomized key generation algorithm $Gen(1^n)$, that takes a security parameter n as input and outputs a pair of keys (pk, sk),
- An encryption algorithm $\mathcal{E} = \{\mathcal{E}_{pk} \mid pk \in \mathcal{K}_{pk}\}$ which may be randomized. It takes a public key and a plaintext as input, and outputs the ciphertext or $\perp$.
- A deterministic decryption algorithm $\mathcal{D} = \{\mathcal{D}_{sk} \mid sk \in \mathcal{K}_{sk}\}$ that takes a private key and a ciphertext as input and outputs the plaintext or $\perp$.

Encryption and Decryption

All algorithms must run in polynomial time. The scheme provides correct decryption, if $\mathcal{D}_{sk}(\mathcal{E}_{pk}(m)) = m$ for each key pair $(pk, sk) \in \mathcal{K}$ and all plaintexts $m \in \mathcal{M}$.

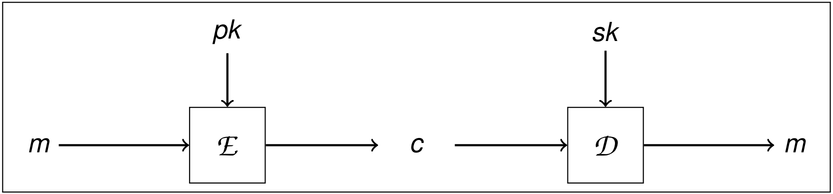


Figure: *Encryption uses a public key pk and decryption a private key sk.*

IND-CPA Security

Adversary		*Challenger*
	$\xleftarrow{\quad pk,\ 1^n \quad}$	$(pk, sk) \xleftarrow{\$} Gen(1^n)$ $b \xleftarrow{\$} \{0,1\}$
Choose m_0, m_1 $\lvert m_0 \rvert = \lvert m_1 \rvert$	$\xrightarrow{\quad m_0,\ m_1 \quad}$	
	$\xleftarrow{\quad c \quad}$	$c \leftarrow \mathcal{E}_{pk}(m_b)$
Select m_0 $(b' = 0)$ or m_1 $(b' = 1)$	$\xrightarrow{\quad b' \quad}$	Compare b and b', output 1 or 0

Public-key EAV and CPA experiment. The adversary can also encrypt any chosen plaintext.

IND-CPA Security

The IND-CPA advantage of the adversary A is defined as

$$\mathrm{Adv}^{\text{ind-cpa}}(A) = |Pr[b' = b] - Pr[b' \neq b]|.$$

The scheme has *indistinguishable encryptions under a chosen plaintext attack* (IND-CPA secure or CPA-secure), if for every probabilistic polynomial time adversary A, the advantage $\mathrm{Adv}^{\text{ind-cpa}}(A)$ is negligible in n.

Since an adversary can encrypt m_0 and m_1 and compare the result with the challenge ciphertext c, it is obvious that a public-key scheme with *deterministic* encryption cannot be IND-CPA secure.

IND-CCA2 Security

A more powerful adversary is able to perform an *adaptive chosen ciphertext attack* (CCA2).

In the CCA2 experiment, the adversary can additionally request the *decryption* of arbitrary ciphertexts (before and after choosing two plaintext messages), except that the challenge ciphertext c cannot be queried.

Plain RSA

Definition

The plain RSA encryption scheme is defined by:

- A polynomial-time key generation algorithm $Gen(1^n)$ that takes the security parameter 1^n as input, generates two random n-bit primes p and q and sets $N = pq$. Furthermore, two integers e and d with $ed \equiv 1 \mod (p-1)(q-1)$ are chosen. $Gen(1^n)$ outputs the public key $pk = (e, N)$ and the private key $sk = (d, N)$.
- The plaintext and the ciphertext space is $\mathbb{Z}_N^*$.
- The deterministic encryption algorithm takes a plaintext $m \in \mathbb{Z}_N^*$ and the public key pk as input and outputs

$$c = \mathcal{E}_{pk}(m) = m^e \mod N.$$

Definition

- The decryption algorithm takes a ciphertext $c \in \mathbb{Z}_N^*$ and the private key *sk* as input and outputs

$$m = \mathcal{D}_{sk}(c) = c^d \mod N.$$

For the correctness of the RSA scheme one has to show that

$$(m^e)^d \equiv m \mod N$$

for all $m \in \mathbb{Z}_N^*$. But this follows from Euler's Theorem: let $m \in \mathbb{Z}_N^*$, then we have

$$m^{\varphi(N)} \equiv 1 \mod N.$$

Since $ed = 1 + k\varphi(N)$ for some $k \in \mathbb{Z}$, we obtain
$m^{ed} = m(m^{\varphi(N)})^k \equiv m \mod N$.

RSA Security

Factoring N breaks RSA, but the opposite statement is not necessarily true. The security of RSA is in fact based on the *RSA assumption* which is stronger than the factoring assumption.

Definition

Consider the following experiment: run the RSA $Gen(1^n)$ algorithm. A uniform ciphertext $c \overset{\$}{\leftarrow} \mathbb{Z}_N^*$ is chosen and an adversary obtains 1^n, e, N and c. The adversary has to find $m \in \mathbb{Z}_N^*$ such that

$$m^e \mod N \equiv c.$$

The RSA problem is hard relative to *Gen*, if for every probabilistic polynomial-time adversary, the probability of finding the correct plaintext m is negligible in n. The *RSA assumption* states that there is a key generation algorithm *Gen* such that the RSA problem is hard.

RSA Pitfalls

The plain RSA encryption scheme is *deterministic* and thus cannot be CPA-secure. But even if the plaintext messages are chosen uniformly at random from a large space, there are a number of pitfalls, e.g.:

- Very small public exponents, e.g., $e = 3$ are insecure (low-exponent attack).
- Small decryption exponents, i.e., $d < \frac{1}{3}N^{1/4}$ are insecure (Wiener attack).
- Partially known plaintexts and keys can be attacked.
- Plain RSA encryption is *malleable* and the ciphertext can be easily manipulated.
- A *chosen ciphertext attack* against plain RSA is easy.

Miller-Rabin Test

The asymptotic density of primes among the first n integer numbers is $\frac{1}{\ln(n)}$. To generate a large prime, choose an *odd random number* of the required size and test its primality.

The Miller-Rabin test uses the fact that a number n with the following property must be *composite*:

(COMP) *There exists a base $a \in \mathbb{N}$ with $1 \leq a < n$, such that*

$$a^d \not\equiv \pm 1 \mod n \text{ and } a^{2^r d} \not\equiv -1 \mod n \text{ for all } 1 \leq r \leq s-1.$$

If (COMP) is satisfied then n *must be* composite. If n does not satisfy (COMP) then n *could be* a prime. The test is repeated several times in order to increase the probability that n is a prime.

Running Time

RSA encryption and decryption requires one exponentiation modulo N. Let $n = \text{size}(N)$. Then the running time is

$$O(n^3).$$

This is polynomial, but not very fast, and a large number of such operations should be avoided.

Often, one takes $e = 2^{16} + 1$, so that the running-time of encryption is only $O(n^2)$. Furthermore, the decryption $c^d \mod N$ can be accelerated by a factor of around 4 by using the *Chinese Remainder Theorem* (CRT). Recall that the CRT gives a decomposition

$$\mathbb{Z}_N \cong \mathbb{Z}_p \times \mathbb{Z}_q$$

if $p \neq q$. Therefore, decryption can done modulo p and modulo q with half the size of N.

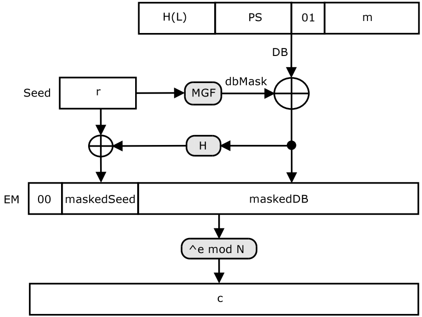
RSA-OAEP

Plain RSA is not CPA-secure, even if the parameters p, q, e, d are appropriately chosen, because the scheme is *deterministic*. Furthermore, the ciphertext is *malleable* and chosen ciphertext attacks are possible. Now, the padded RSA scheme *Optimal Asymmetric Encryption Padding* (OAEP) is standardized in PKCS #1 version 2.2 and RFC 8017.

The message m is transformed into an encoded message EM (see below), which is subsequently encrypted using plain RSA.

Decryption first computes EM, verifies the integrity of the result and then recovers m.

RSA-OAEP



Encryption of a plaintext m using RSA-OAEP. MGF is a mask generating function which is similar to a hash function, but has variable output length.

A major result is that RSA-OAEP is *secure against adaptive chosen ciphertexts attacks* (CCA2-secure) under the RSA assumption and in the random oracle model.

Factoring Methods

Factoring is assumed to be a hard problem, at least on conventional computers. Obviously, if the factoring assumption turns out to be wrong, then RSA is broken.

Trial division is an elementary factoring method. It suffices to test numbers $\leq \sqrt{N}$. A list of small primes is useful (sieve method), and otherwise all odd numbers (or perhaps all numbers not divisible by 2, 3 or 5) need to be tested. The worst-case complexity is $O(\sqrt{N})$ and the running time is exponential in size (N).

Pollard's ρ

Pollard's ρ *algorithm* searches for an integer x such that $\gcd(x, N)$ is either p or q, e.g., $x \equiv 0 \mod p$, but $x \not\equiv 0 \mod q$. The idea is to generate a pseudorandom sequence $x_i = f(x_{i-1})$ of integers modulo N and to find a collision modulo p or q using Floyd's cycle finding algorithm. Note that

$$x_k \equiv x_{2k} \mod p \Longleftrightarrow x_k - x_{2k} \equiv 0 \mod p.$$

The algorithm computes pairs x_k, x_{2k} of integers modulo N and checks whether $\gcd(x_k - x_{2k}, N) > 1$.

By the birthday paradox, around $O(\sqrt{p}) = O(N^{1/4})$ iterations should be sufficient to find a collision modulo p.

Fermat Factorization

Fermat factorization uses a representation of N as a difference of squares:

$$N = x^2 - y^2 = (x+y)(x-y)$$

To find x and y, you begin with the integer $x = \lceil \sqrt{N} \rceil$ and increase x by 1 until $x^2 - N$ is square, say y^2, so that $N = x^2 - y^2$. Fermat factorization always works, since N can be written as a difference of two squares:

$$pq = \left(\frac{1}{2}(p+q)\right)^2 - \left(\frac{1}{2}(p-q)\right)^2 = x^2 - y^2$$

However, Fermat's method is only efficient if the prime factors are close to one another, i.e., if y is small. In general, the running time is $O(\sqrt{N})$.

Quadratic Sieve

The *quadratic sieve* generalizes Fermat factorization and is currently the fastest algorithm for numbers with less than around 100 decimal digits. One looks for integers x and y such that $x^2 \equiv y^2 \mod N$, but $x \not\equiv \pm y \mod N$. This implies

$$N \text{ divides } x^2 - y^2 = (x+y)(x-y),$$

but N divides neither $x+y$ nor $x-y$. Hence $gcd(x-y, N)$ must be a non-trivial divisor of N and equals either p or q. The idea is to multiply several (non-quadratic) numbers $x^2 - N$ with small prime factors (*smooth over a factor base*). The difficult task is to find smooth numbers.

The running time of the quadratic sieve is *sub-exponential*:

$$O(e^{(1+o(1))\sqrt{\ln(N)\ \ln(\ln(N))}}).$$

Quadratic Sieve Example

Let $N = 10441$, then $\sqrt{N} \approx 102.2$. We compute $x^2 - N$ for a couple of integers $x \geq 103$ and factorize the result.

```
x=103  168 = 2^3 * 3 * 7
x=104  375 = 3 * 5^3
x=105  584 = 2^3 * 73
x=106  795 = 3 * 5 * 53
x=107  1008 = 2^4 * 3^2 * 7
x=108  1223 = 1223
x=109  1440 = 2^5 * 3^2 * 5
```

$$(103^2 - N) \cdot (104^2 - N) \cdot (107^2 - N) \cdot (109^2 - N) = 2^{12} \cdot 3^6 \cdot 5^4 \cdot 7^2$$

Now we set $x = 103 \cdot 104 \cdot 107 \cdot 109$ and $y = 2^6 \cdot 3^3 \cdot 5^2 \cdot 7$ and obtain $x^2 \equiv y^2 \mod N$. We have $x \equiv 7491$, $y \equiv 10052$ modulo N and get $\gcd(7491 - 10052, 10441) = 197$, which is in fact a divisor of 10441.

Number Field Sieve

At the time of writing, the *number field sieve* is the most efficient algorithm for factoring large integers. With massive computing resources, numbers with more than 200 digits and for example the *RSA Challenge* with 768 bits could be factored using this method.

The heuristic complexity of the number field sieve is

$$O(e^{(c+o(1))\ln(N)^{1/3}\ \ln(\ln(N))^{2/3}}),$$

where $c = \sqrt[3]{\frac{64}{9}} \approx 1.92$.

Other Methods

Pollard's $p-1$ method can be applied if $p-1$ or $q-1$ decompose into a product of small primes. In this case, one can guess multiples k of $p-1$. We have

$$(p-1) \mid k \Rightarrow a^k \equiv 1 \mod p.$$

Hence $\gcd(a^k-1, N)$ gives either p (method successful) or N (failure).

The *Elliptic Curve Factorization method* (ECM) is another interesting factoring method with sub-exponential running time. ECM is suitable for finding prime factors with up to 80 decimal digits. For larger primes, the quadratic sieve or the number field sieve are more efficient.

Furthermore, *Shor's algorithm* can efficiently factor large numbers on a quantum computer. However, sufficiently large and stable quantum computers are not yet available.