NTNU | Norwegian University of Science and Technology

PUBLIC KEY CRYPTOGRAPHY AND THE RSA SCHEME

TTM4135 - Lecture 8

Tjerand Silde

29.01.2026

Motivation

- Public key cryptography (PKC) provides some features which cannot be achieved with symmetric key cryptography (which requires shared keys)
- PKC is widely applied for key management (key exchange) and authentication (signatures) in protocols such as TLS and IPSec
- RSA is probably the best known public key cryptosystem, widely deployed in many kinds of applications, and can be used to provide both

Contents

One-way functions

- A function f is said to be a *one-way function* if it is easy to compute $f(x)$ given x, but is computationally hard to compute $f^{-1}(y) = x$ given y
- It is an open problem in computer science whether any one-way functions formally exist, but we have many candidates that we trust and use
- Some examples of functions believed to be one-way are:
  1. Multiplication of large primes: the inverse function is integer factorization
  2. Exponentiation: the inverse function is taking discrete logarithms
  3. Hashing: breaking the one-wayness seem heuristically hard

Trapdoor one-way functions

- A *trapdoor one-way function* f is a one-way function such that given additional information (the trapdoor) it is easy to compute f^{-1}
- An example of a trapdoor one-way function is *modular squaring*
- Let $n = pq$ be the product of two large prime numbers p and q and define $f(x) = x^2 \bmod n$ for some integer $0 < x < n$
- If there is an algorithm to take square roots (compute f^{-1}) then this algorithm can be used to factorize n
- The trapdoor is the factorization of n – knowledge of p and q gives an efficient algorithm to find square roots (exercise)

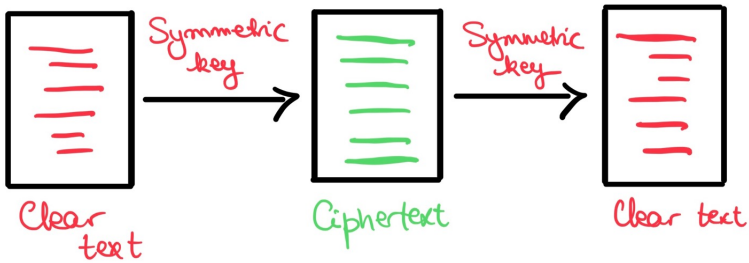
Ciphers based on computationally hard problems

- In 1976 Diffie and Hellman published their famous paper *New Directions in Cryptography*
- They suggested that computational complexity be applied in the design of encryption algorithms
- A public key system can be designed by using a trapdoor one-way function
- The trapdoor will become the decryption key

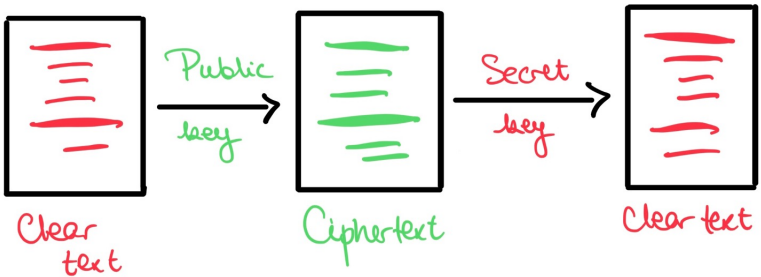
Public and private keys

- Public key cryptography is another name for *asymmetric cryptography*
- The encryption and decryption keys and algorithms are different
- The encryption key is a *public key* which can be known to anybody
- The decryption key is a *private key* which should be known only to the owner of the key (the only secret in the scheme)
- Finding the private key from knowledge of the public key must be a hard computational problem

Symmetric-key Cryptography – reminder



Public-key Cryptography – reminder



Why public key cryptography?

- Public key cryptography has two main advantages in comparison with shared key (symmetric key) cryptography:

1. The key management is simplified: keys do not need to be transported confidentially but can be agreed upon over an untrusted network.
2. Digital signatures can be obtained to ensure the authenticity and integrity of a message. We look at digital signatures in a later lecture.

Using public key encryption

- In a public key encryption scheme the receiver's key is made public
- Suppose that user A stores her public key, PK_A, in a public directory
- Anyone can obtain this public key and use it to encrypt a message M for A as following: $C = E(M, PK_A)$
- Since only A has the private key, SK_A, only A can decrypt and recover the message: $M = D(C, SK_A)$

Hybrid encryption

- Public key cryptography is usually larger and computationally much more expensive than symmetric-key cryptography
- A typical usage of public key cryptography is to:
  - encrypt a random key for a symmetric-key encryption algorithm
  - encrypt the message M using the symmetric-key algorithm
- This can be done in the following way:
  1. B chooses a random symmetric key k, finds A's public key PK_A and computes $C_1 = E(k, PK_A)$
  2. B computes $C_2 = E_s(M, k)$ where E_s is encryption with a symmetric-key algorithm, such as AES in CTR mode
  3. B sends (C_1, C_2) to A
- On receipt of (C_1, C_2), A recovers $k = D(C_1, SK_A)$ and then $M = D_s(C_2, k)$

Contents

Introduction to RSA

- Rivest-Shamir-Adleman, MIT, 1977
- Public-key encryption and digital signature scheme
- Based on integer factorization problem
- RSA patent expired in 2000

RSA Key Generation

1. Let p, q be distinct prime numbers, randomly chosen from the set of all prime numbers of a certain size
2. Compute the product $n = pq$
3. Select e randomly with $\gcd(e, \phi(n)) = 1$
4. Compute $d = e^{-1} \bmod \phi(n)$
5. The public key is the pair n and e
6. The private key consists of the values p, q and d

RSA operations

Encryption The public key for encryption is $K_E = (n, e)$

1. Input is any value M where $0 < M < n$
2. Compute $C = E(M, K_E) = M^e \bmod n$

Decryption The private key for decryption is $K_D = d$
(we will see later how to use values p and q)

1. Compute $D(C, K_D) = C^d \bmod n = M$

Note that any message needs to be pre-processed to become the input M: this includes coding as a number and adding randomness (details later)

Numerical example

- ***Key Generation***:
  - Suppose $p = 43, q = 59$ then $n = pq = 2537$ and $\phi(n) = (p-1)(q-1) = 2436$
  - Choose $e = 5$ then $d = e^{-1} \bmod \phi(n) = 5^{-1} \bmod 2436 = 1949$
- ***Encryption***: Let $M = 50 \implies C = M^5 \bmod 2537 = 2488$
- ***Decryption***: Then $M = C^{1949} \bmod 2537 = 50$

Correctness of RSA Encryption

We need to know that encryption followed by decryption gets back where we started from:

$$(M^e)^d \bmod n = M$$

Since $d = e^{-1} \bmod \phi(n)$ we know that $ed \bmod \phi(n) = 1$ and so $ed = 1 + k\phi(n)$ for some integer k. Therefore:

$$\begin{aligned}(M^e)^d \bmod n &= M^{ed} \bmod n \\ &= M^{1+k\phi(n)} \bmod n\end{aligned}$$

To complete the proof we need to show

$$M^{1+k\phi(n)} \bmod n = M \tag{1}$$

Proving RSA correctness: Case 1

There are two cases. We first assume $\gcd(M, n) = 1$
We can apply Euler's theorem directly to get

$$M^{\phi(n)} \bmod n = 1$$

Therefore

$$\begin{aligned} M^{1+k\phi(n)} \bmod n &= M \times (M^{\phi(n)})^k \bmod n \\ &= M \times (1)^k \bmod n \\ &= M \end{aligned}$$

Proving RSA correctness: Case 2

- If $\gcd(M, n) \neq 1$ then either $\gcd(M, p) = 1$ or $\gcd(M, q) = 1$
- Suppose that $\gcd(M, p) = 1$ (the other case is similar).
  Then $\gcd(M, q) = q$ so $M = lq$ for some integer l
- Applying Fermat's theorem we obtain

$$(M^{\phi(n)})^k \bmod p = (M^{(p-1)})^{(q-1)k} \bmod p = 1^{(q-1)k} \bmod p = 1$$

  Therefore

$$M^{1+k\phi(n)} \bmod p = M \bmod p \tag{2}$$

- Since $M = lq$ it follows that

$$M^{1+k\phi(n)} \bmod q = 0 \tag{3}$$

Case 2 continued

- Finally the Chinese Remainder Theorem tells us that there is a unique solution $x \bmod n$ to the two equations (2) and (3) where $x = M^{1+k\phi(n)}$
- The solution $x = M$ satisfies both equations (2) and (3) and therefore this is the unique solution for $M^{1+k\phi(n)} \bmod n$
- Thus the RSA equation is satisfied in this case too

RSA applications

The RSA operations can be used in a variety of applications:

- In this lecture we consider only message encryption.
- We look at RSA digital signatures in a later lecture.
- RSA is often used to distribute a key for symmetric-key encryption (often known as *hybrid encryption*).
- RSA can be used for user authentication by proving knowledge of the private key corresponding to an authenticated public key.

Contents

Implementation issues

Optimizations in the implementation of RSA have been widely studied. We examine some of the most important issues:

- key generation
  - choice of e
  - generating large primes
- encryption and decryption algorithms
  - fast exponentiation
  - using CRT for decryption
- formatting data (padding)

Generating p and q

- The primes p and q should be random of a chosen length. Today this length is usually recommended to be at least 1024 bits.

- A simple method of selecting a random prime is given as following:
  1. Select a random odd number r of the required length.
  2. Check whether r is prime
  3.
     - If so, output r and halt
     - If not, increment r by 2 and go to the previous step.
  4. We require a fast way to check for primality such as Miller–Rabin.

Are there enough prime numbers?

- The *prime number theorem* tells us that the primes thin out as the numbers get larger.
- Let $\pi(x)$ denote the number of prime numbers less than x. The prime number theorem say that the ratio of $\pi(x)$ and $\frac{x}{\ln(x)}$ tends to 1 as x grows.
- We can use the prime number theorem to give a rule of thumb that the proportion of prime numbers up to size x is $\ln(x)$.
- Since $\ln(2^{1024}) = 710$ then one in every 710 numbers of size 1024 bits is a prime number. Therefore there are well over 2^{1000} 1024-bit primes.
- Thus brute-force searching for randomly chosen primes is infeasible.

Selecting e

- The public exponent e should be chosen at random for best security
- A small value of e is often used in practice since it is more efficient.
  - $e = 3$ is the smallest possible value and is sometimes used. However, there are possibly security problems when encrypting small messages.
  - $e = 2^{16} + 1$ is a popular choice. More exponentiations, but reduces the constraints on p and q, and avoids aforementioned attacks.
- A smaller than average d value is also possible. However, to avoid known attacks d should be at least $\sqrt{n}$ (and if it is too small we can brute-force it)

Fast exponentiation

- To compute the RSA encryption and decryption functions we use the *square-and-multiply* modular exponentiation algorithm.
- We write e in binary representation.

$$e = e_0 2^0 + e_1 2^1 + \cdots + e_k 2^k$$

  where e_i are bits.

- The idea behind fast exponentiation is the *square-and-multiply* algorithm.
- There are many variants and optimisations of the basic idea.

Square-and-multiply algorithm

Require: $m, n, e = e_k e_{k-1} \ldots e_1 e_0$ with $e_k = 1$
Ensure: $m^e \bmod n$

1: $z \leftarrow m$
2: **for** $i = k - 1$ **downto** 0 **do**
3: $\quad z \leftarrow z^2 \bmod n$
4: $\quad$ **if** $e_i = 1$ **then**
5: $\quad\quad z \leftarrow (z \cdot m) \bmod n$
6: $\quad$ **end if**
7: **end for**
8: **return** z

Cost of square and multiply

- If $2^k \leq e < 2^{k+1}$ then the algorithm uses k squarings. If b of the e_i bits are 1 then the algorithm uses $b - 1$ multiplications. Note that the first computation $z \rightarrow z \cdot m$ is not counted because then $z = 1$.
- Suppose that n is a 2048-bit RSA modulus. The public exponent e is of length at most 2048 bits. To compute $M^e \bmod n$ requires at most:
  - 2048 modular squarings; and
  - 2048 modular multiplications.
- For a randomly chosen exponent roughly half of the bits of e are '1' bits and so only 1024 multiplications are needed on average.
- Remember that we can reduce modulo n after every operation.

Faster decryption with the CRT

- We can use the Chinese Remainder Theorem to decrypt ciphertext C faster with regard to p and q separately.
- First compute:

$$\begin{aligned} M_p &= C^{d \bmod p-1} \bmod p \\ M_q &= C^{d \bmod q-1} \bmod q \end{aligned}$$

- Solve for $M \bmod n$ using the Chinese remainder theorem.

$$\begin{aligned} M &\equiv M_p \pmod{p} \\ M &\equiv M_q \pmod{q} \end{aligned}$$

$$M = q \times (q^{-1} \bmod p) \times M_p + p \times (p^{-1} \bmod q) \times M_q \bmod n$$

Why it works

Note that $d = d \bmod (p-1) + k(p-1)$ for some k.

$$\begin{aligned} M \bmod p &= (C^d \bmod n) \bmod p \\ &= C^d \bmod p \\ &= C^{d \bmod p-1} C^{k(p-1)} \bmod p \\ &= C^{d \bmod p-1} \\ &= M_p \end{aligned}$$

- Similarly $M \bmod q = M_q$
- Therefore $M \bmod n$ is the unique solution to the above two equations.

Example

- Same example as before: $n = 43 \times 59$, the ciphertext is $C = 2488$ and the decryption exponent is $d = 1949$.
- $d \bmod p - 1 = 1949 \bmod 42 = 17$ and $d \bmod q - 1 = 1949 \bmod 58 = 35$
- This gives us:

$$\begin{aligned} M_p &\equiv 2488^{17} \pmod{43} = 37^{17} \pmod{43} = 7 \\ M_q &\equiv 2488^{35} \pmod{59} = 16^{35} \pmod{59} = 50 \end{aligned}$$

- Using CRT the solution is $M = 50$.

How much faster is decryption with the CRT?

- Note that the exponents ($d \bmod p-1$) and ($d \bmod q-1$) are about half the length of d.
- Since the complexity of exponentiation (square and multiply) increases with the cube of the input length, computing M_p and M_q each use 1/8 of the computation of computing $M = C^d \bmod n$.
- Overall there is about 4 times less computation. If M_p and M_q can be computed in parallel the time can be up to 8 times faster.
- This is a good reason to store p and q with the private exponent d.

RSA Padding

- Using the RSA encryption function directly on messages encoded as numbers is a weak cryptosystem. It is vulnerable to attacks such as:
  - building up a dictionary of known plaintexts
  - guessing the plaintext and checking to see if it encrypts to the ciphertext
  - Håstad's attack (next slide)
- Hence, padding mechanisms must be used for messages being encrypted. These mechanisms must include redundancy and randomness.

Håstad's Attack

- Suppose that the *same message* is encrypted without padding to three different recipients.
- Suppose that public exponent $e = 3$ is used by all recipients.
- Then the cryptanalyst has three ciphertexts:

$$\begin{aligned} c_1 &= m^3 \bmod n_1 \\ c_2 &= m^3 \bmod n_2 \\ c_3 &= m^3 \bmod n_3 \end{aligned}$$

- These equations can be solved by the CRT to obtain m^3 in the ordinary (non-modular) integers. Then m can be found by taking a cube root.

Example RSA block format: PKCS Number 1

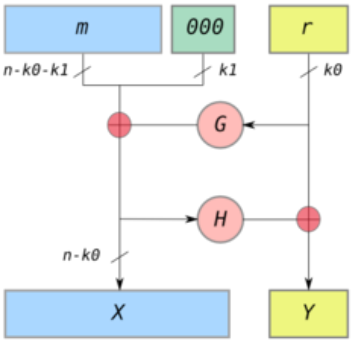
Encryption block format is:

00	02	*PS*	00	*D*

where 00 and 02 are bytes, *PS* is a pseudo-random string of nonzero bytes, and *D* is the data to be encrypted.

- The length of the block is the same as the modulus.
- PS is a minimum of 8 bytes, but more if the data D is small.
- The byte 02 and padding ensure that even short messages result in a large integer value for encryption.

Optimal Asymmetric Encryption Padding (OAEP)



Picture from Wikipedia

- The OAEP scheme includes k_0 bits of randomness and k_1 bits of redundancy into the message before encryption.
- Reasonable values of k_0 and k_1 are 128.
- Two hash functions G and H are used.
- Note that OAEP is an encoding algorithm - it can be easily inverted without any secret.

Contents

Factorizing the RSA modulus

- If an adversary can factorize the modulus n into its prime factors p and q the adversary can easily recover the private key d and reveal all messages. Thus breaking RSA is not harder than the factorization problem.
- Using a formal definition of encryption security it can be shown that breaking RSA is as hard as the so-called *RSA problem*.
- It is unknown whether the RSA problem is as hard as the factorization problem. Remember that it is also unknown whether factorization is really computationally hard.
- One positive security result is that finding the private key from the public key is as hard as factorizing the modulus.

Equivalence with factorization problem

- Is it possible to find the private key without factorizing the modulus? No!

Theorem (Miller)

Determining d from e and n is as hard as factorizing n.

- To show this, suppose that a cryptanalyst can find d from e and n.
- Then cryptanalyst could factorize n using Miller's algorithm (next slide).
- Algorithm uses same ideas as Miller–Rabin test for primality.

Miller's Algorithm

- Define u, v such that $ed - 1 = 2^v u$, where u is odd
- Consider the sequence $a^u, a^{2u}, \ldots, a^{2^{v-1}u}, a^{2^v u} \pmod{n}$, where a is random with $0 < a < n$.
- Notice that $a^{2^v u} = a^{ed-1} = a^{ed}a^{-1} = aa^{-1} = 1 \pmod{n}$. Therefore there is a square root of 1 somewhere in this sequence.
- With probability at least $\frac{1}{2}$ the sequence contains a non-trivial square root of 1 modulo n, thereby revealing the factors of n.
- If not, choose a new a and repeat.

Side Channel Attacks

Sometimes we can break a scheme based on how it is implemented instead of breaking the underlying mathematics.

Many different kinds of side-channels are now known including:

Timing attacks Uses timing of the private key operations to obtain information about the private key.

Power analysis Uses power usage profile of the private key operations to obtain information about the private key.

Fault analysis Measures the effect of interfering with the private key operations to obtain information about the private key.

Timing attacks

- Recall that the square-and-multiply algorithm performs either a squaring or a squaring and a multiplication in each step
- The multiplication step is included exactly when each exponent bit $e_i = 1$
- Thus step i takes around twice as long when $e_i = 1$ as when $e_i = 0$

Some side-channel countermeasures

- Computing in constant time - run a “dummy” multiplication when $e_i = 0$
- Montgomery ladder - makes every operation depend on the key to avoid some fault attacks while keeping the computation constant time
- Randomizing the RSA message - mitigates “differential” attacks by preventing multiple timings on the same operation

Questions?