Vietnam National University Ho Chi Minh City, University of Science
Department of Information Technology

Asymmetric cipher & Digital signature

Assoc. Prof. Trần Minh Triết
PhD. Trương Toàn Thịnh

fit@hcmus

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Introduction

- The problem that arises in conventional cryptosystems is the sharing of the key *k* between sender *A* and receiver *B*.
- In fact, the need to change the contents of the key *k* is necessary, so the exchange of information about the key *k* between *A* and *B* is necessary.
- To secure the key *k*, *A* and *B* must communicate with each other over a truly secure and secret communication channel.
- However, it is very difficult to guarantee the security of the communication channel, so the key *k* can still be detected by person *C*!

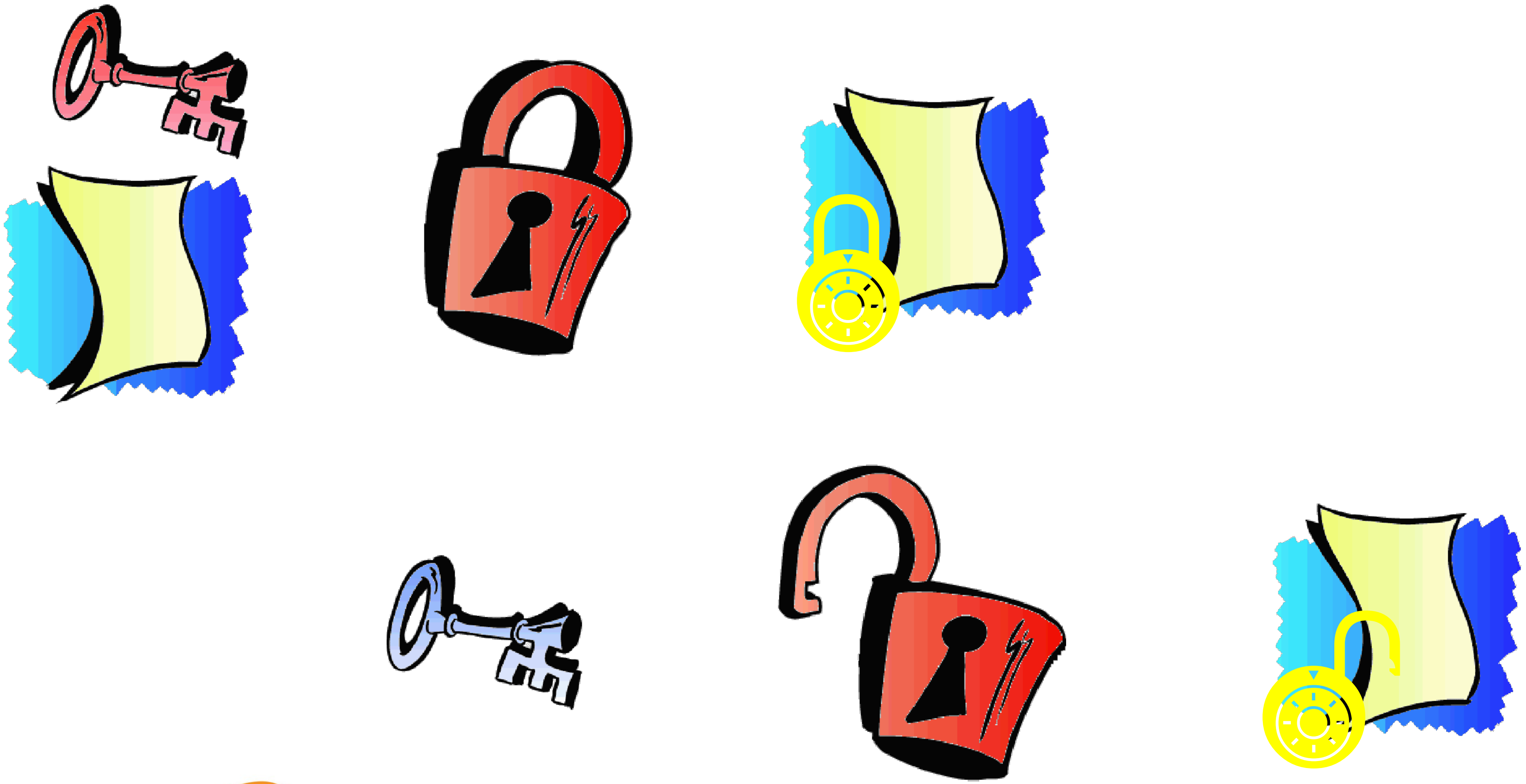
Introduction

- The idea of a public key cryptosystem was introduced in 1976 by Martin Hellman, Ralph Merkle and Whitfield Diffie at Stanford University.
- Subsequently, the Diffie-Hellman method of Martin Hellman and Whitfield Diffie was published.
- In 1977, in the journal "The Scientific American", the authors Ronald Rivest, Adi Shamir and Leonard Adleman published the RSA method, the famous public key encryption method and widely used today in applications of encryption and protect information

Introduction

- A public-key system uses 2 types of keys in the same key pair:
  - Public key is widely available and used in information encryption
  - Private key is held by only one person and is used to decrypt information encrypted with the public key.
- These coding methods exploit *f* mappings for which the back mapping f^{-1} is very difficult compared to performing the *f*.
- Only when the private key is known can the reverse mapping f^{-1} be performed.

Public-key cryptosystem



cdio 4.0
fit@hcmus

RSA

- In 1978, R.L. Rivest, A. Shamir and L. Adleman proposed the RSA public key encryption system (also known as the "MIT system"). In this method, all calculations are performed on $\mathbb{Z}_n$ where n is the product of two distinct odd primes p and q.
- So, we have ㄹㅎ$(n) = (p - 1)$ ㅎ $(q - 1)$
  - Let $P = C = Z_n$ and define: $K = \{((n, p, q, a, b)$: $n = p$ ㅎ q, p, q are distinct prime numbers, a ㅎ b ㄹㄱ 1 (mod ㄹㅎ$(n))\}$
  - For each $k = (n, p, q, a, b)$ ㅍ K, let:
    - $e_k(x) = x^b$ mod n and $d_k(y) = y^a$ mod n, for x, y ㅍ Z_n
  - n and b are published (public-key)
  - p, q, a are kept secret (private-key)

RSA

- Generate two prime numbers with large values p and q
- Compute $n = p$ ㅎ q and ㄹㅎ$(n) = (p - 1)$ ㅎ $(q - 1)$
- Randomly choose an integer b ($1 < b <$ ㄹㅎ(n)) such that gcd(b, ㄹㅎ(n)) = 1
- Compute $a = b^{-1}$ mod ㄹㅎ(n) (using extended Euclide algorithm)
- n and b are published (public-key)
- p, q, a are kept secret (private-key)

Some attacks on RSA

- The security nature of the RSA method is based on the fact that the cost of invalidating encrypted information will be too great to be considered impossible.
- Since the key is public, RSA cracking attacks often rely on the public key to determine the corresponding private key. It is important to rely on n to calculate p, q of n, from which d can be calculated.

fit@hcmus

Using ㄹㅎ(n)

- Suppose the attacker knows the value ㄹㅎ(n). Then the determination of the value p, q is taken to solve the following two equations: $n = p$ ㅎ q
- Let $q = n/p$, we have a quadratic equation: ㄹㅎ$(n) = (p - 1)$ ㅎ $(q - 1)$ and $p^2 - (n -$ ㄹㅎ$(n) + 1)$ ㅎ $p + n = 0$
- p, q are the two solutions of this quadratic equation. However, the problem of detecting the value ㄹㅎ(n) more difficult than determining two prime factors of n.

fit@hcmus

Using factorization algorithm p-1

- Algorithm (input n and B)
  - $a = 2$
  - **for** $j = 2$ **to** B **do** $a = a^j \bmod n$ //$a^{2 \times 3 \times \cdots \times B} = a^{B!}$
  - $d = gcd(a - 1, n)$
  - **if** $1 < d < n$ **then** d is a prime factor of n (success)
  - **else** cannot determine the prime factor of n (failure)
- Algorithm Pollard p - 1 (1974) is one of the efficient simple algorithms used to factorize large integers. The input parameter of the algorithm is an (odd) integer n that needs to be factored into prime factors and the limit value B.
- Assume $n = p \times q$ (p, q unknown) and B is a large enough integer, for each prime factor $k \le B \rightarrow k \mid (p - 1) \Rightarrow (p - 1) \mid B!$

cdio 4.0

fit@hcmus

Using factorization algorithm p-1

- Analysis
  - At the end of the loop (step 2): a ㄹㄱ $2^{B!}$ (mod n) ㄷ a ㄹㄱ $2^{B!}$ (mod p)
  - Due to $p|n$, Fermat theorem allows: $2^{p\text{-}1}$ ㄹㄱ 1 (mod p)
  - Due to (p - 1) | B!, at step 3 of algorithm: a ㄹㄱ 1 (mod p)
  - At step 4: p | (a˜ 1) and p | n, if $d = gcd(a$˜ 1, $n)$ then $d = p$
- Example
  - Assume n = 79523 (= 281 ㅎ 283)
    - Using algorithm p–1 with B = 8, determine a = 51705 & $d = gcd(a – 1, n)$ = 281
    - In this case, prime factorization is successful due to value 281 – 1 = 280 = 2^3 ㅎ 5 ㅎ 7 ㄷ 280ㄹㅅB!
  - Assume n = 76693 (= 271 ㅎ 283)
    - Using algorithm p–1 with B = 8, determine a = 71515 & $d = gcd(a – 1, n)$ = 1
    - In this case, prime factorization failed due to the value 271 – 1 = 270 = 2 ㅎ 3^3 ㅎ 5 ㄷ 270 B!

Using factorization algorithm *p*-1

- In algorithm $\tilde{p}$ 1, we have $\tilde{B}$ 1 exponentiations of modulo, each requires a maximum 2 ㅎ $\log_2 B$ modulo multiplication using squaring and multiplying algorithms
- Computing GCD using extended Euclidean algorithm with complexity $O((\log_2 n)^3)$.
- So, algorithmic complexity is $O(B$ ㅎ $\log_2 B(\log_2 n)^2 + (\log_2 n)^3)$
- The probability of choosing the value *B* is relatively small and satisfying the condition is very low.
- When increasing the value of *B* (such as *B* ㄹㅁ) then the algorithm will succeed, but this algorithm will not be faster than the divisibility algorithm as shown above.

Using factorization algorithm p-1

- This algorithm is only effective when attacking the RSA method in the case that n has a prime factor p that (p - 1) has only very small prime divisors.
- We can easily construct an RSA public-key cryptosystem that is secure against the attack algorithm p - 1. The simplest way is to find a large prime p_1, which $p = 2p_1 + 1$ is also prime numbers, similarly find q_1 large prime and $q = 2q_1 + 1$ prime.

fit@hcmus

Key-break based on repeat attacks

- Simons and Norris: RSA systems can be vulnerable to repeated attacks. If the adversary knows public key-pair $\{n, e\}$ and cipher-text C, it can compute the sequence of the following:

$$\left.\begin{array}{l} C_1 = C^e \pmod{n} \\ C_2 = C_1^{\,e} \pmod{n} \\ \ldots \\ C_i = C_{i\text{-}1}^{\,e} \pmod{n} \end{array}\right\} \quad \textit{next } C = (\textit{previous } C)^e$$

- If there is a C_j in $C_1, C_2, C_3, \ldots, C_i$ such that $C_j = C$, then we have $M = C_{j\text{-}1}$ because $C_j = C_{j\text{-}1}^{\,e} \pmod{n}$ and $C = M^e \pmod{n}$
- Example: if attacker knows $\{n, e, C\} = \{35, 17, 3\}$, it computes
  - $C_1 = C^e \pmod{n} = 317 \pmod{35} = 33$
  - $C_2 = C_1^{\,e} \pmod{n} = 3317 \pmod{35} = 3$
  - Because $C_2 = C \Rightarrow M = C_1 = 33$

Information hiding in the RSA system

- *RSA* is characterized by information that is not always concealed.
- Assume sender sends $e = 17$, $n = 35$. If it wants to send any data in {1, 6, 7, 8, 13, 14, 15, 20, 21, 22, 27, 28, 29, 34}, then result of encryption is the same as original data. So, $M = M^e \bmod n$.
- If $p = 109$, $q = 97$, $e = 865$ then RSA there is absolutely no hiding information, becausẹ M, $M = M^{865} \bmod (109$ ㅎ $97)$
- For each value of n, there are at least 9 cases where the resulting encoding is the original source data.
- We have, $M = M^e \bmod n$, or: $M = M^e \bmod p$ & $M = M^e \bmod q$ (*)
  - For each e, each equality in (*) have at least 3 solutions ㅍ {0, 1, -1}.
  - Number of unmasked messages (not changed after encryption): $m = [1 + gcd(e - 1, p - 1)][1 + gcd(e - 1), q - 1]$

cdio 4.0
fit@hcmus

Comments

- The key to being able to decrypt the information is to get the *p* and *q* values that make up the *n* value.
- Given these 2 values, we can calculate ㄹㅎ$(n) = (p - 1)$ ㅎ $(q - 1)$ & $d = e^{-1}$ mod ㄹㅎ(n) according to extended Euclidean algorithm.
- If the integer *n* can be factored, i.e., the values of *p* and *q* can be determined, then the security of the RSA method is no longer guaranteed.
- Thus, the security of the RSA method based on computers at the present time is not capable of solving the factorization of very large integers.
- In 1994, Peter Shor, a scientist at AT&T labs, came up with an algorithm that could efficiently parse very large integers on a quantum computer.

cdio 4.0
fit@hcmus

Prime number problem

- To ensure the security of the RSA system, the integer $n = p \times q$ must be large enough that the factorization of n cannot be easily performed. Currently, prime factorization algorithms can handle integers over 250 decimal digits, or 829 binary digits (*bits*).
- To be safe, the primes p and q need to be large enough, for example, over 125 digits. The problem here is to solve the problem: how to quickly and accurately check whether a positive integer n is prime or composite?
- By definition, a positive integer n is prime –| n only divisible by 1 and n (consider only positive integers). So, n is prime number –| n has no positive divisors ᄑ [2, …,]
- So, n is prime number –! i ᄑ [2, …,], ᅰ|(n ° 0 mod i)

https://en.wikipedia.org/wiki/RSA_numbers

Miller-Rabin algorithm

- Verify that a positive integer *n* is prime by the above method will give correct results.
- However, the processing time of the algorithm is obviously very large, or even impossible, in the case of relatively large *n*.
- In fact, verifying that a positive integer *n* is prime often applies methods belonging to the Monte Carlo group of algorithms
- Example:
  - Algorithm Solovay-Strassen
  - Algorithm Miller-Rabin

Miller-Rabin algorithm

- Monte Carlo algorithm group
  - Used in the affirmation or negation of something. The algorithm always gives an answer and the answer obtained is only likely to be “Yes” or “No”.
  - The “yes-biased Monte Carlo” algorithm is a Monte Carlo algorithm in which the answer “Yes” is always correct but the answer “No” may not be correct.
- Miller-Rabin . Algorithm
  - Advantages of fast processing (positive integer n is checked in time proportional to $log_2 n$, i.e., a number of bits in the n’s binary representation)
  - It is possible that the algorithm's conclusion is not completely correct, that is, there is a chance that a composite number n will be concluded to be prime, although the probability of the incorrect conclusion is not high.
  - Overcome by executing the algorithm many times to reduce the possibility of false conclusions below the allowable threshold ⬭ highly reliable conclusions

Miller-Rabin algorithm

- The idea is based on the following two theorems
- Fermat's little theorem:
  - n is prime number $\Rightarrow a^{n-1} \equiv 1 \bmod n$ $(1 < a < n)$
  - Example $n = 561$, we have $5^{560} \equiv 1 \bmod 561$ (But $561 = 3 \times 11 \times 17$)
    - Call 561 is a Fermat pseudo-primes with base 5
  - Example $n = 341$, we have $2^{340} \equiv 1 \bmod 341$ (But $341 = 11 \times 31$)
    - Call 341 is a Fermat pseudo-primes with base 2
  - Clearly, inverting this theorem may not be correct.
  - However: $a^{n-1} \not\equiv 1 \bmod n$ $(1 < a < n) \Rightarrow n$ is composite (Correct)
- If n is prime $\Rightarrow$ '1' has 2 solutions modulo n: '1' & '$n - 1$'
  - Note: writing '-1' standing for '$n - 1$'

fit@hcmus

Miller-Rabin algorithm

- Suppose we need to check if n is prime or not?
  - Analyze $n = m$ ᄒ $2^k + 1$ (m odd) ⊣ $n - 1 = m$ ᄒ 2^k
  - Randomly choose $1 < a < n - 1$ (Why not choose 1 and $n - 1$?)
  - Consider: , , , …,,
    - If a^m ° ⩵ ⩵ 1 mod n ⩵ ° a^{n-1} ° 1 mod n (n is prime and stop)
    - If a^m 1 mod n & a^m n - 1 mod n
      - Try $(a^m)^2$ ° $n - 1$ mod n (mod n, due to a^m 1 mod n & a^m n - 1 mod n)
        - If correct ⩵ ° a^{n-1} ° 1 mod n (n is prime and stop)
      - Try $((a^m)^2)^2$ ° $n - 1$ mod n (mod n, due to $a^{m ᄒ 2}$ 1 mod n & $a^{m ᄒ 2}$ n - 1 mod n)
        - If correct ⩵ ° a^{n-1} ° 1 mod n (n is prime and stop)
      - …
      - Try ° $n - 1$ mod n (mod n, due to 1 mod n & n - 1 mod n)
        - If correct ⩵ ° a^{n-1} ° 1 mod n (n is prime and stop)
        - If incorrect ⩵ n is composite (Correct 100%)

Miller-Rabin algorithm

- Description: positive integer analysis $n = 2^k m + 1$ with m odd
  - Randomly choose a positive integer $a \in \{1, 2, ..., n - 1\}$
  - Compute $b = a^m \bmod p$
  - if $b \equiv 1 \pmod p$ then "p is prime" and stop
  - for $i = 0$ to $k - 1$
    - if $b \equiv p - 1 \pmod p$ then "p is prime" and stop
    - else $b = b^2 \bmod p$
  - Conclude "p is composite"
- Comments:
  - A "yes-biased Monte Carlo" algorithm for the statement "positive integer n is composite".
  - The probability of the wrong conclusion, i.e., the algorithm making the conclusion "n is prime" when n is indeed composite, is only 25%.
  - If we apply the algorithm k times with different values of a and we still get the conclusion "n is prime", then the correct probability of this conclusion is $1 - 4^{-k} \approx 1$, with large-enough k.

Arithmetic processing

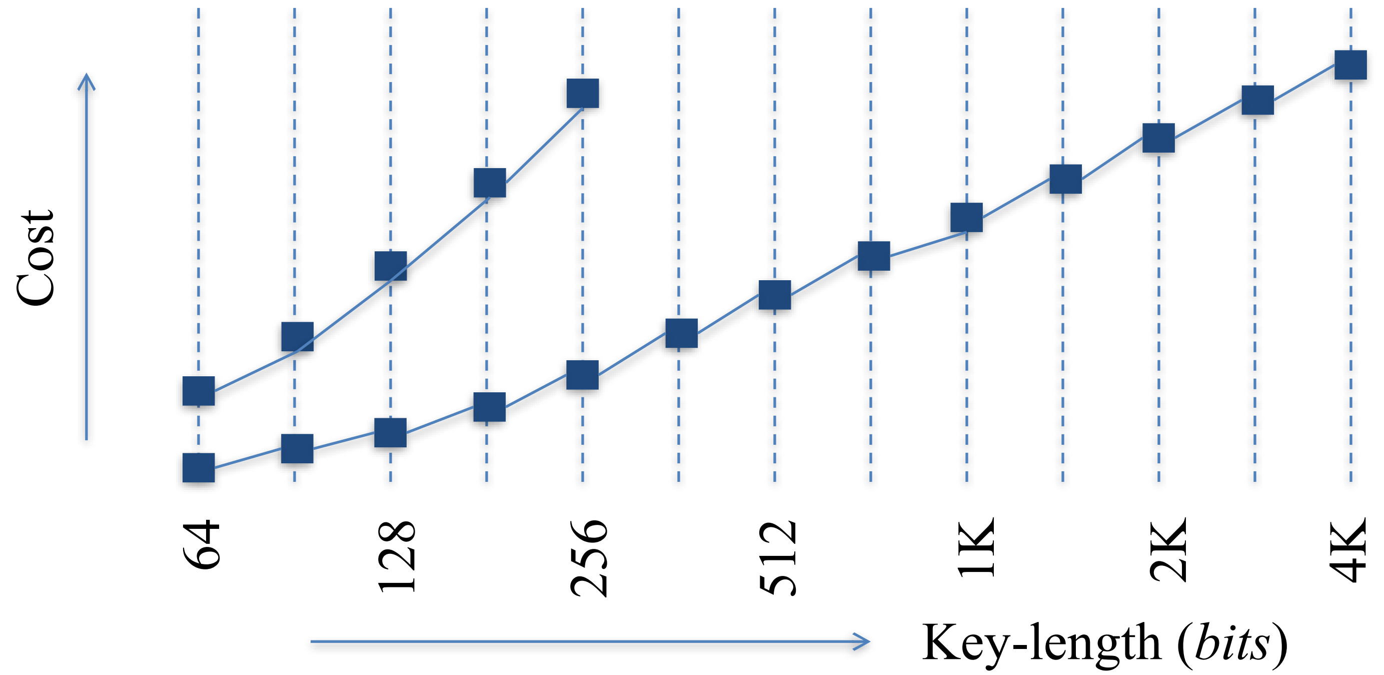
- Compute $z = x^b \bmod n$
- Algorithm “squared and multiplied”
  - Let b be $b_{l-1}b_{l-2}\ldots b_1b_0$, b_i ㅍ $\{0, 1\}$, 0 ㅞ $i < l$
  - $z = 1$
  - $x = x \bmod n$
  - **for** $i = l - 1$ **down to** 0
    - $z = z^2 \bmod n$
    - **if** $b_i = 1$ **then** $z = z$ ㅎ $x \bmod n$

Symmetry vs asymmetry

- Conventional encryption methods have the advantage of being very fast compared to public key encryption methods.
  - Because the key used for encryption is also used for decryption, it is necessary to keep the contents of the key and the key secret, called the secret key. Even in the case of a direct exchange of keys, the key is still detectable. The difficult problem for these encryption methods is the key exchange problem.
- The public key is more vulnerable than the secret-key.
  - To find the secret key, the decryptor needs some more information related to the characteristics of the source text before encrypting to find the decryption clue instead of having to use the key extraction method.
  - Determining whether the message after decryption is indeed the original message before encryption is again a difficult problem.
  - For public keys, cracking is completely possible provided there are enough resources and processing time.

fit@hcmus

Symmetry vs asymmetry



Graph comparing the cost of breaking secret-keys & public-keys

Contents

Introduction

- Goals of digital signature:
  - User authentication
  - Data integrity
  - Non-repudiation
- Basic concepts:
  - Digital signature: data string that allows to identify the origin/entity that generated a message.
  - Algorithm for generating digital signatures: methods for generating digital signatures
  - Digital signature strategy includes ***algorithm generating digital signature*** and ***algorithm verifying digital signature***.
  - **Digital signature scheme = Digital signature generation algorithm + Digital signature verification algorithm**

Introduction

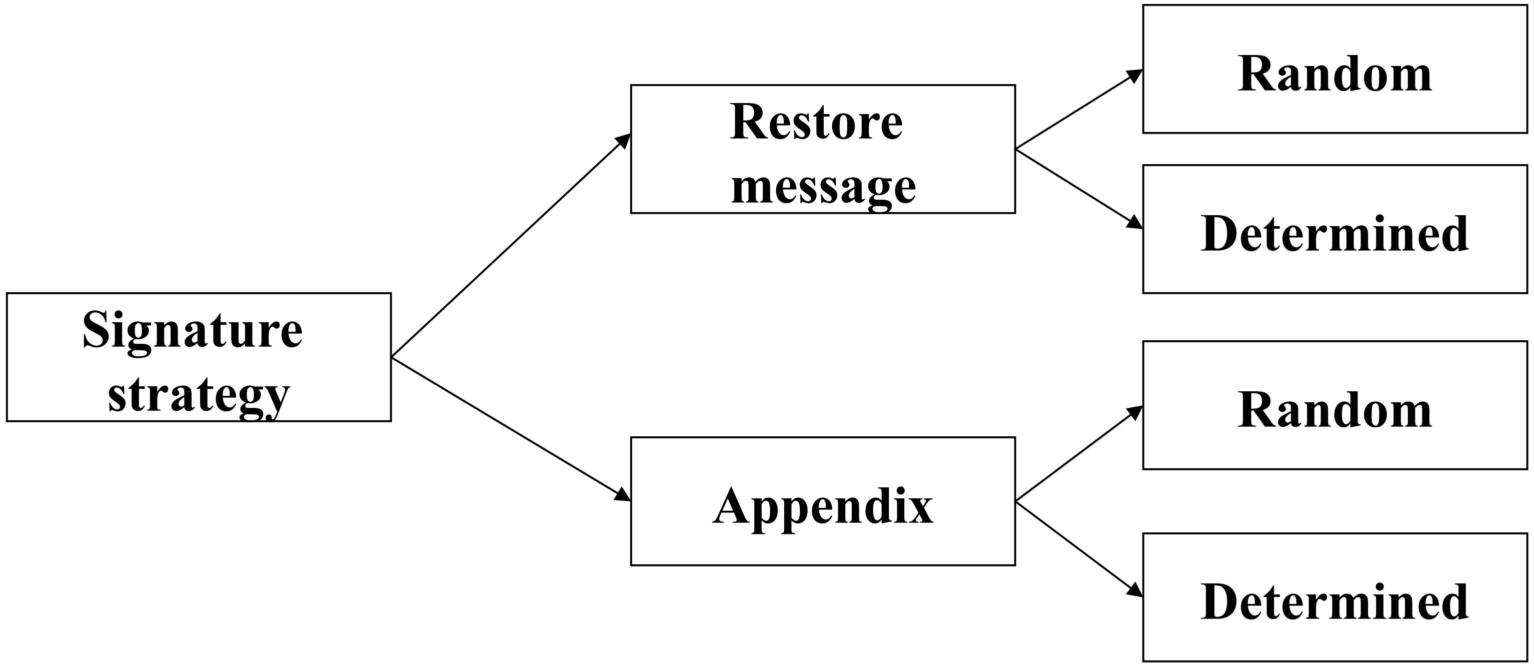
- Notations:

M	Message-space
M_S	Signed-message space
S	Signature-space
R	Map 1-1 from M to M_S (redundancy function)
M_R	Image of R
R^{-1}	Inverse function of R
h	One-way function with a source set M
M_h	Hash-value space (h: $M \to M_h$)

fit@hcmus

Introduction

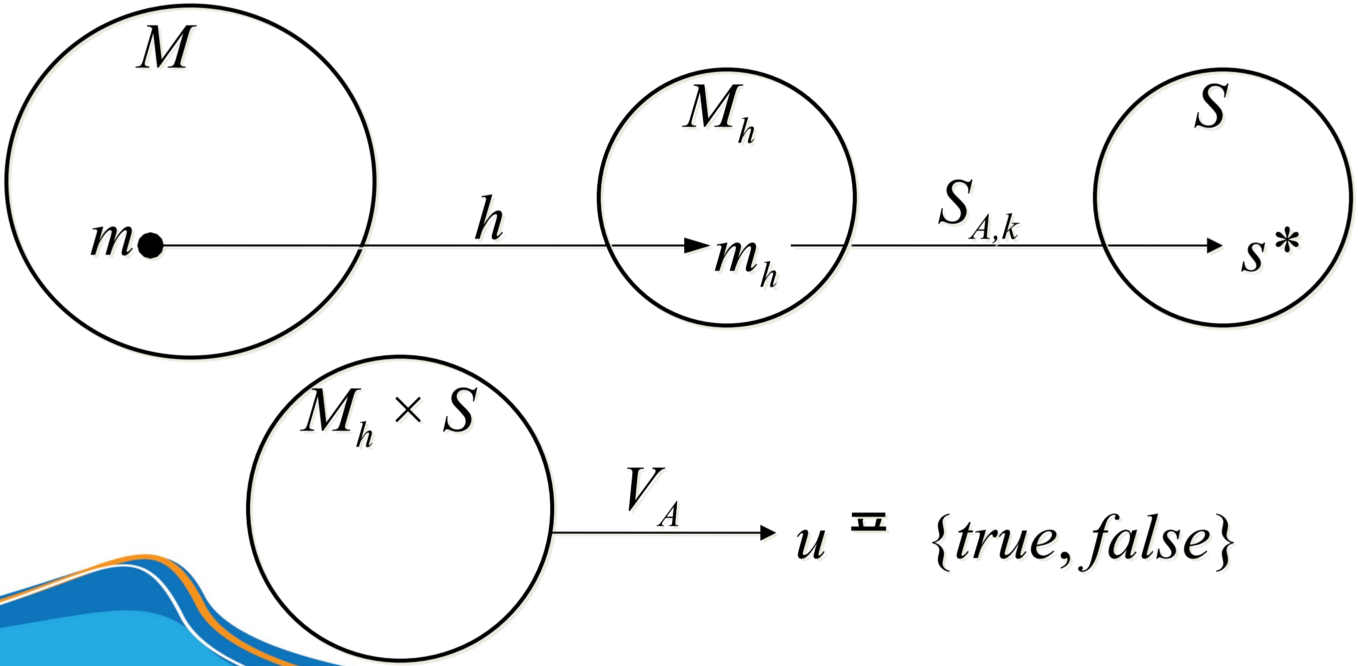
- Classification of digital signatures



$$s^* = S_{A,k}(m_h)$$
$$u = V_A(m_h, s^*)$$

Introduction

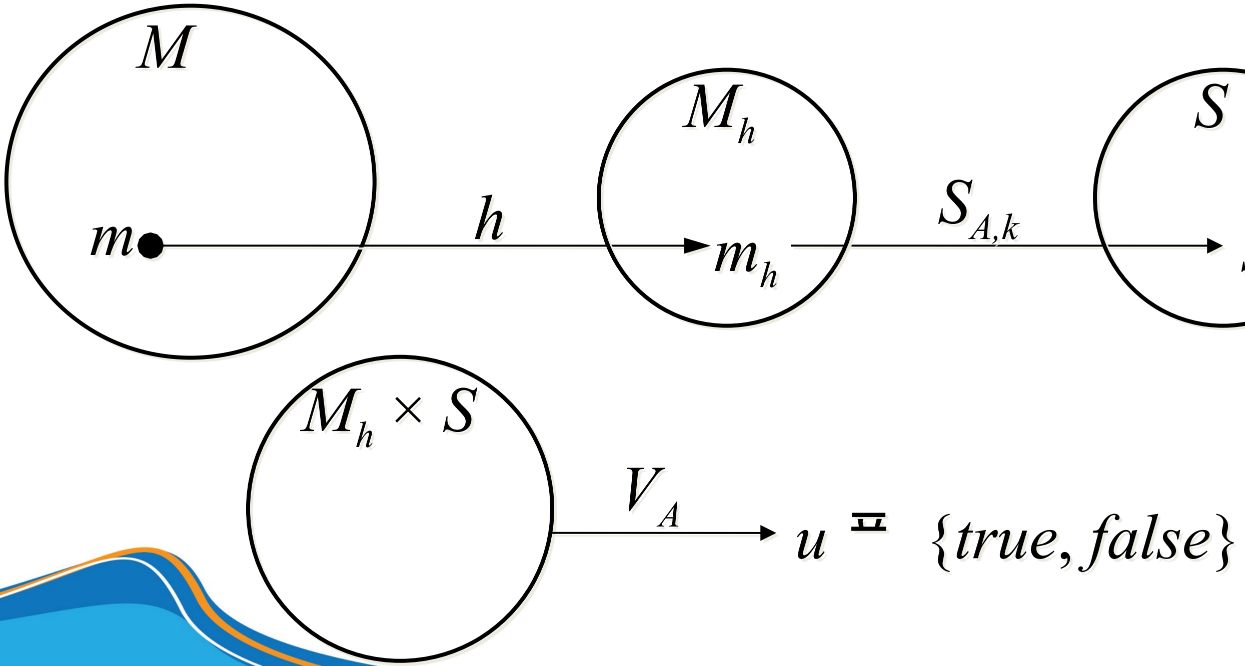
- Signature strategies with appendix
  - Signature accompanies original message (Ex: DSA, ElGamal, Schnorr…)
  - Message (original) required for digital signature verification
  - Use a cryptographic hash function (instead of a redundancy function)



Introduction

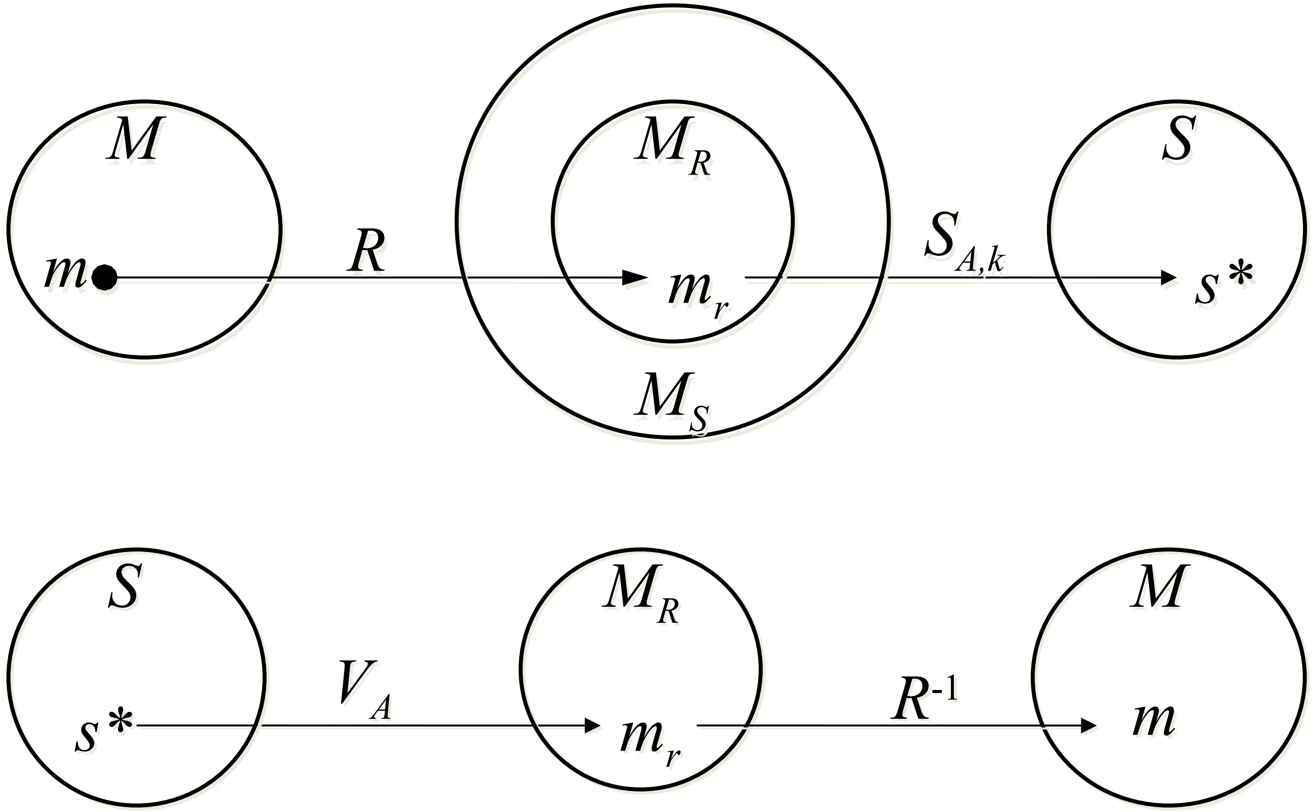
$$s^* = S_{A,k}(m_h)$$
$$u = V_A(m_h, s^*)$$

- Requirements:
  - For each $k \in$ R, easy to compute $S_{A,k}$
  - Easy to compute V_A
  - Hard for those are not *signer* to find $m \in M$ and $s^* \in S$ such that $V_A(m', s^*) = true$, for $m' = h(m)$



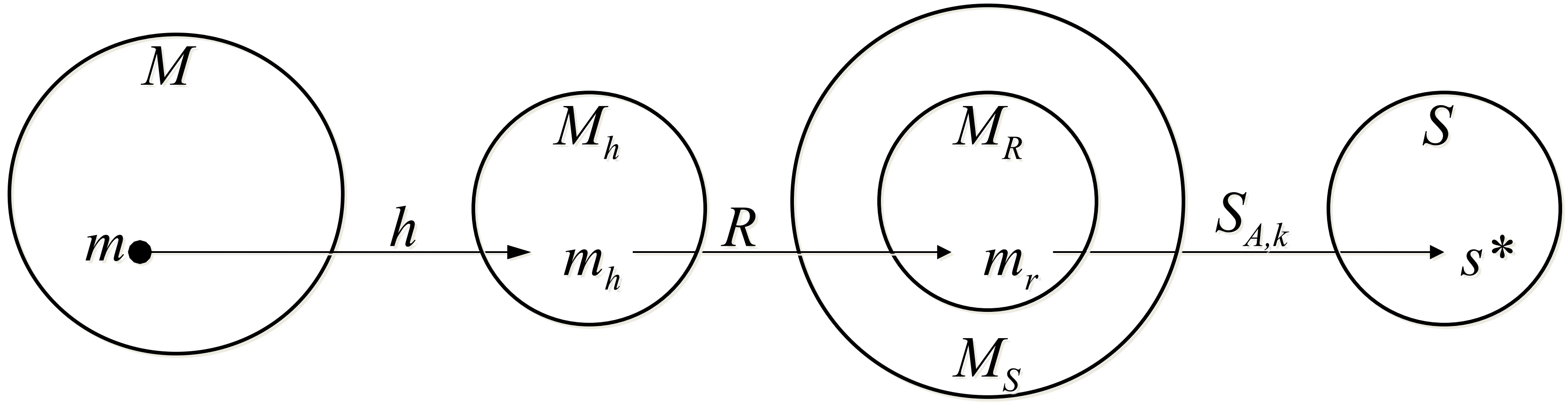
Introduction

- A digital signature can allow the message to be recovered

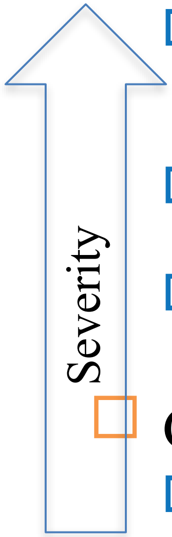


Introduction

- Requirements:
  - For each k ㅍ R, easy to compute $S_{A,k}$
  - easy to compute V_A
  - Hard for (computationally infeasible) those are not A to find s^* ㅍ S such that $V_A(s^*)$ ㅍ M_R



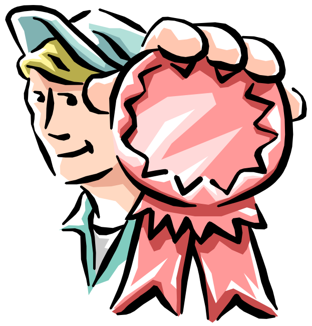
Introduction



- Levels of "Breaking" the e-signature strategy:
  - ***Total Break***: find an efficient way to "fake" a valid signature.
    - Know private key?
    - Do not know private-key but find an efficient way to create a valid signature.
  - ***Selective forgery***: Given a message, attacker *can* create a valid signature for this message.
  - ***Existential forgery***: can find and point to a message (maybe nonsense) but easy for attacker to create valid signature for this message.
- Classification of attacks
  - ***Key-only***: attacker **only know public-key**
  - Message attack
    - ***Known-message attack***: attacker **has the signatures of a set of messages**. Attacker **knows contents of these messages** but **is not allowed to pre-choose** these messages.
    - ***Chosen-message attack***: attacker **has valid signatures** of **a set of selective messages** (non-adaptive)
    - ***Adaptive chosen-message attack***: attacker can use signer as a "**oracle**"

RSA signature

- Generate key n, p, q, e, d
- Create signature
  - Compute $m_r = R(m)$
  - Compute $s = m_r^{\mathrm{d}} \bmod n$
  - Signature of m is s
- Verify signature
  - Receive public-key (n, e)
  - Compute $m_r = s^e \bmod n$
  - Verify m_r ㅍ M_r
  - Recover $m = R^{-1}(m_r)$



RSA signature

- Attack
  - Factorize a large integer
  - Possibility of multiple key pairs yielding the same signature
  - Homomorphic property:

    $E(x_1) \times E(x_2) = x_1^e \times x_2^e \bmod n = (x_1 \times x_2)^e \bmod n = E(x_1 \times x_2 \bmod n)$
- Re-blocking problem
- Importance of redundancy function: ISO/IEC 9796
- Efficiency (p, q are k-bit prime-numbers): cost of generating signature $O(k^3)$ and verifying signature $O(k^2)$
- Bandwidth
  - Bandwidth depends on R.
  - Example: ISO/IEC 9796 maps a k-bit message to a $2k$-bit string in M_S with $2k$-bit signature.

fit@hcmus

DSA signature

- Generating key:
  - Choose 1 prime number q 160 bits
  - Choose 0 ㅞ t ㅞ 8, choose $2^{511 + 64t} < p < 2^{512 + 64t}$ for $q \mid p - 1$
  - Choose g in Z_p^*, and ㄱㅅ $= g^{(p - 1) / q} \bmod p$, ㄱㅅ ㄴㅎ 1 (ㄱㅅ is a primitive of sub-group with order q of Z_p^*)
  - Choose 1 ㅞ a ㅞ q - 1, compute $y =$ ㄱㅅ$^a \bmod p$
  - Public-key (p, q, ㄱㅅ, y) and private-key a
- Create signature:
  - Randomly choose an integer k, $0 < k < q$
  - Compute $r =$ (ㄱㅅ$^k \bmod p$) mod q and $k^{-1} \bmod q$
  - Compute $s = k^{-1}$ ㅎ $(h(m) + a$ ㅎ $r) \bmod q$
  - Signature = (r, s)

DSA signature

- Verifying signature
  - Check $0 < r < q$ and $0 < s < q$, if this doesn’t hold, signature isn’t valid
  - Compute $w = s^{-1} \bmod q$ and $h(m)$
  - Compute $u_1 = w$ ㅎ $h(m) \bmod q$, $u_2 = r$ ㅎ $w \bmod q$
  - Compute $v = ($ ㄱㅅ^{u_1} ㅎ $y^{u_2} \bmod p) \bmod q$
  - Signature is valid ―| $v = r$
  - Explain:

    $h(m)$ ㄷ° $-a \times r + k \times s \pmod q$

―| $w \times h(m) + a \times r \times w$ ° $k \pmod q$

―| $u_1 + a \times u_2$ ° $k \pmod q$

―| $\bmod p \pmod q$ ° ㄱㅅ$^{k} \bmod p \pmod q$

DSA signature

- Safety issues of DSA: discrete logarithm in Z_P^* and cyclic sub-group with order q
- Parameters: $q \sim 160$ bits, $p \sim 768$ bits $\sim$ 1Kb
- Failure probability: During verification, necessary to calculate the inverse of s. If $s = 0$ there doesn't exist inverse $\mathbf{Pr}[s = 0] = ()^{160}$
- Efficiency
  - Create signature
    - A modulo exponentiation operation and several operations 160-bit (if $p \sim 768$ bit)
    - The exponentiation can be calculated in advance
    - ***Faster than RSA***
  - Check signature
    - Two operations of exponentiation modulo
    - ***Slower than RSA***

ElGamal signature

- Generate key: p, q, ㄱㅅ, a, $y =$ ㄱㅅ^{a} mod p
  - ㄱㅅ is a primitive of Z^*_p
  - Public-key (p, ㄱㅅ), private-key (a)
- Create signature
  - Randomly choose k, 1 ㅞ k ㅞ p - 1, $gcd(k, p$ - 1) = 1
  - Compute $r =$ ㄱㅅ^{k} mod p, k^{-1} mod (p-1) and $s = k^{-1}$ ㅎ ($h(m)$ - a ㅎ r) mod (p-1)
  - Signature (r, s)
- Check signature
  - Check 1 ㅞ r ㅞ p – 1
  - Compute $v_1 = y^r$ ㅎ r^s mod p, $h(m)$ and $v_2 =$ ㄱㅅ$^{h(m)}$ mod p
  - Signature is valid ㅓ $v_1 = v_2$

Explain

s ㅇ $k^{-1}\times(h(m) - a \times r)$ (mod $p - 1$)

ㅓ $k \times s$ ㅇ $h(m) - a \times r$ (mod $p - 1$)

ㅓ (ㄱㅅ^{a})$^{r} \times r^{s}$ mod p

ElGamal signature

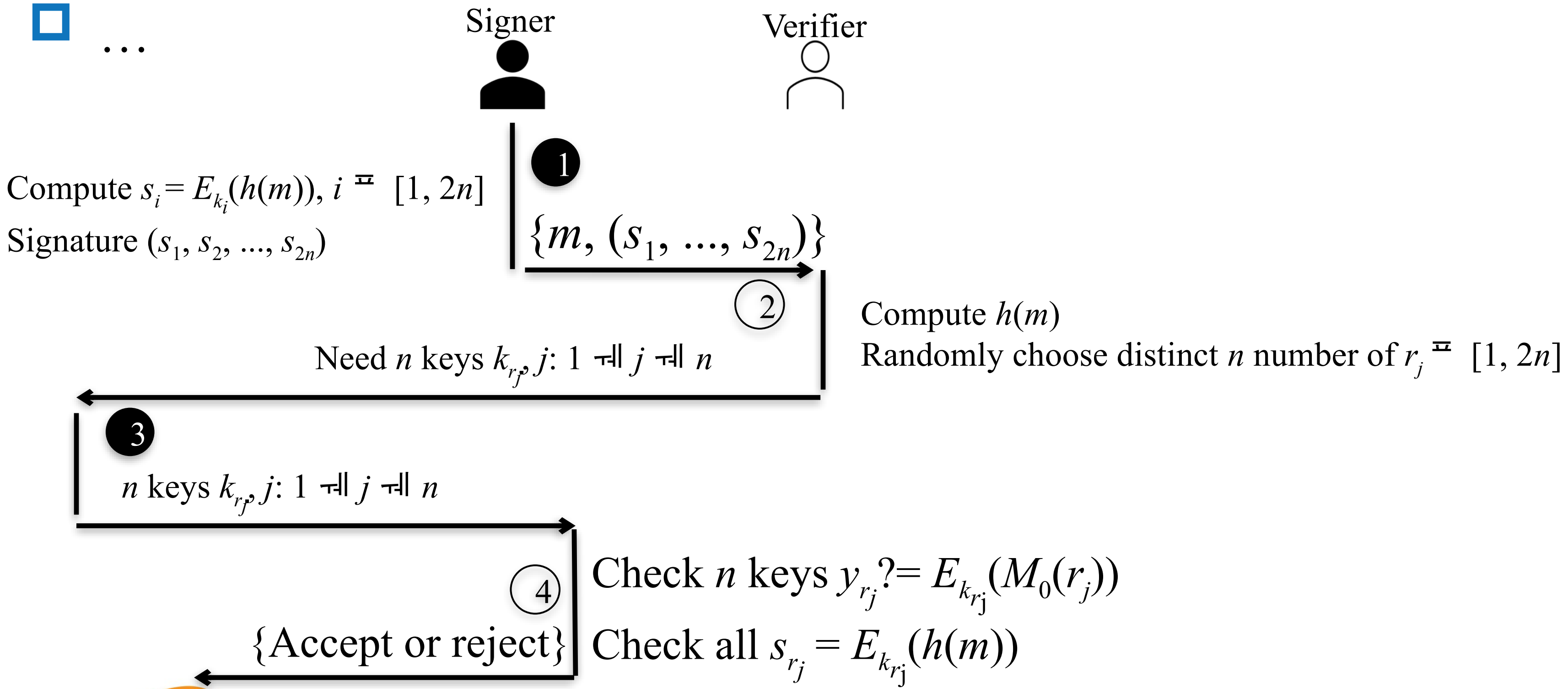
- Problems
  - Value k must be distinguished for each signed message
    - $(s_1 - s_2) \equiv k = (h(m_1) - h(m_2)) \bmod (p - 1)$
    - If $gcd((s_1 - s_2), p - 1) = 1$ then can determine the value k, and have private-key a
  - If you don’t use the hash function, you may get an existential forgery
- Efficiency
  - Create signature
    - An exponentiation operation modulo
    - An operation that uses the Euclidean algorithm to compute the inverse
    - Two multiplication operations modulo
  - Check signature: three exponentiation operations modulo
- Read more: Generalized ElGamal Signature

One-time signature (Rabin method)

- Definition: the signature strategy is used to **sign up to one message**
  - If reused, can be hacked to create fake signatures.
  - A new public key is required for each message to be signed
- Most one-time signature strategies are characterized by very fast signature generation and verification
- Create key: choose symmetric encryption algorithm E (ex: DES)
  - Generating $2n$ random string $k_1, k_2 \ldots, k_{2n} \in K$, each has length of l
  - Compute $y_i = E_{k_i}(M_0(i)), i \in [1, 2n])$
    - $M_0(i)$ is the binary representation of i filled with more bits 0 at the beginning to get the sequence l-bit
  - Public key is $(y_1, y_2, \ldots y_{2n})$ and private key is $(k_1, k_2, \ldots k_{2n})$

One-time signature (Rabin method)

- If there is a conflict, there is a need Trusted Third Party (*TTP*)
  - Signer may deliberately deny having created the signature
  - Verifier may deliberately not accept the signature
  - …



Compute $s_i = E_{k_i}(h(m))$, i ㅍ $[1, 2n]$

Signature $(s_1, s_2, ..., s_{2n})$

$\{m, (s_1, ..., s_{2n})\}$

Compute $h(m)$

Randomly choose distinct n number of r_j ㅍ $[1, 2n]$

Need n keys k_{r_j}, j: 1 ᅰ j ᅰ n

n keys k_{r_j}, j: 1 ᅰ j ᅰ n

Check n keys y_{r_j} ?= $E_{k_{r_j}}(M_0(r_j))$

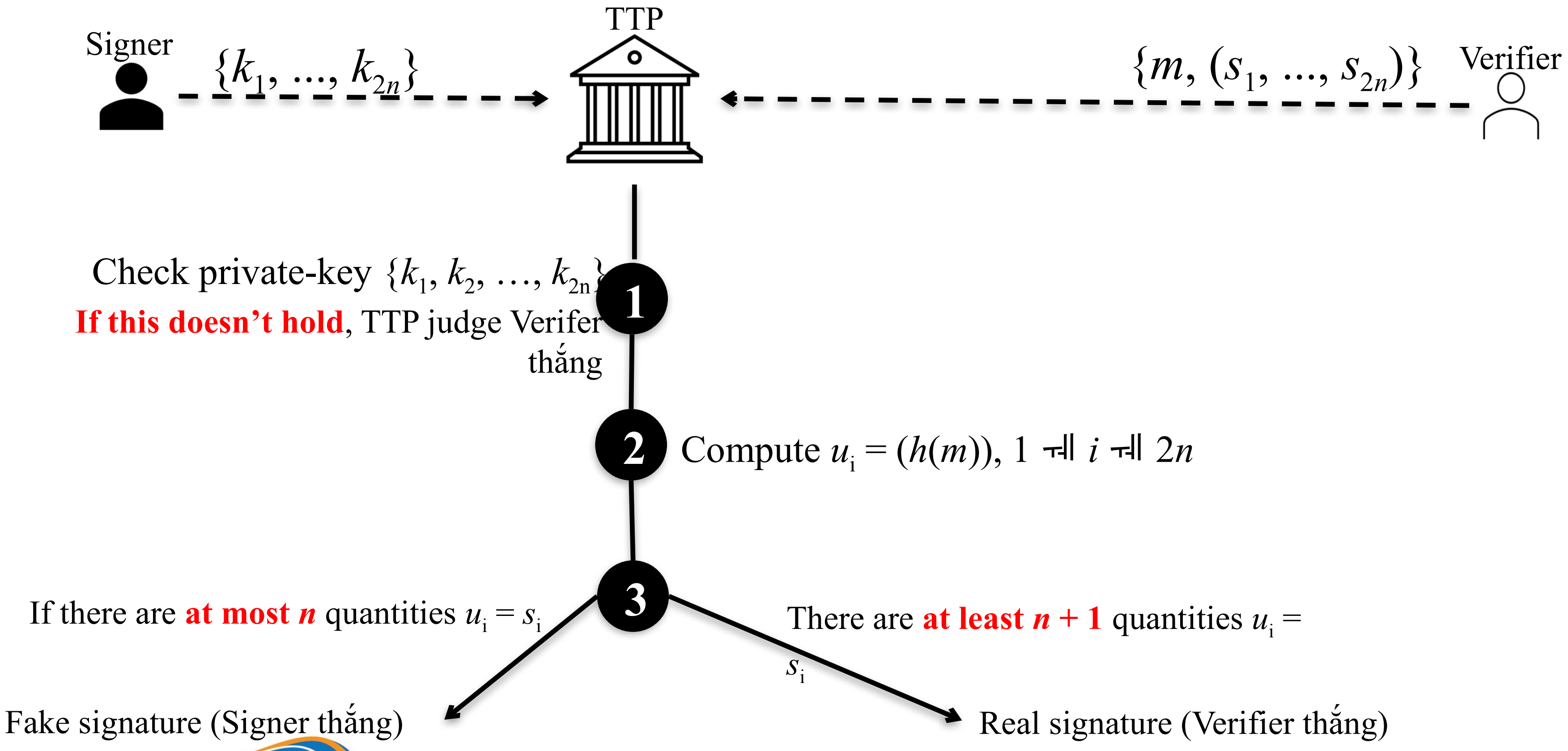
Check all $s_{r_j} = E_{k_{r_j}}(h(m))$

{Accept or reject}

fit@hcmus

One-time signature (Rabin method)

- Context: Signer refuse to generate signature provided by Verifier



Check private-key $\{k_1, k_2, \ldots, k_{2n}\}$
If this doesn't hold, TTP judge Verifer thắng

Compute $u_i = (h(m))$, $1 \le i \le 2n$

If there are **at most n** quantities $u_i = s_i$

Fake signature (Signer thắng)

There are **at least $n + 1$** quantities $u_i = s_i$

Real signature (Verifier thắng)