Week 7: Modern Asymmetric Ciphers

PhD. Ngoc-Tu Nguyen

tunn@uit.edu.vn

Outline

- Why asymmetric cryptography?
- Factoring Based Cryptography (P1)
- Logarithm Based Cryptography (P2)
  - ElGamal cipher;
  - Diffie-Hellman key exchange;
- Elliptic Curve Cryptography (P3)
- Some advanced cryptography system (quantum resistance)

Publick key: https://www.vietcombank.com.vn ???

Why Public-Key Cryptosystems?

To overcome two of the most difficult problems associated with symmetric encryption:

- **Key distribution (key for sysmetric encryption)**
  - How to have secure communications in general without having to trust a KDC with your key
- **Digital signatures**
  - How to verify that a message comes intact from the claimed sender

Whitfield Diffie and Martin Hellman: proposed a method that addressed both problems (1976)

Moden Asymmetric ciphers

Symmetric cipher vs Asymmetric cipher

Hommophic, Searchable encryption,..

$ID_A(atributes)$ $ID_B(atributes)$

$\Updownarrow$

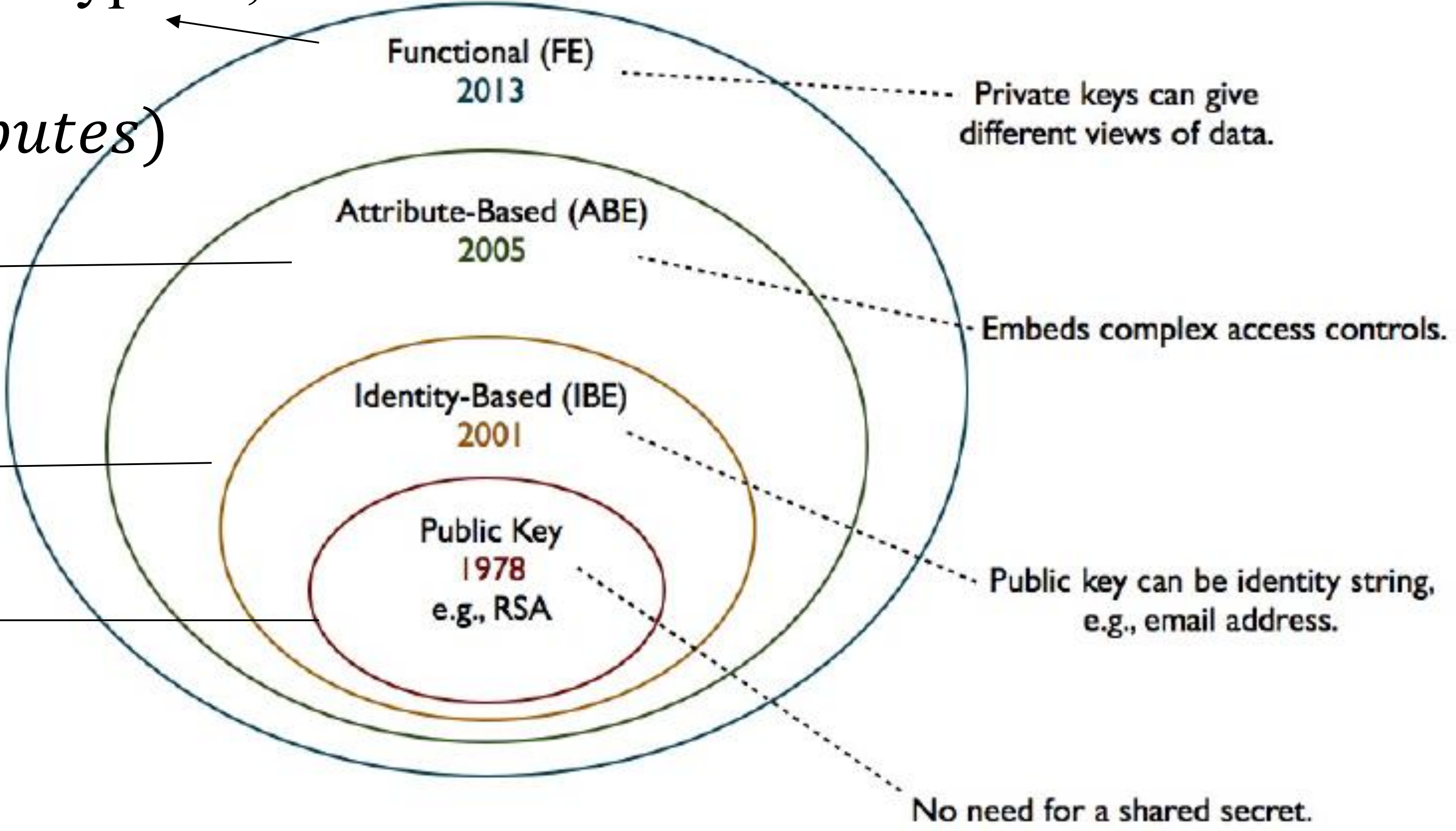
$(PK_A, \{SK_A, SK_B, ..\})$

ID

$\Updownarrow$

(PK_A, SK_A)

(PK_A, SK_A)

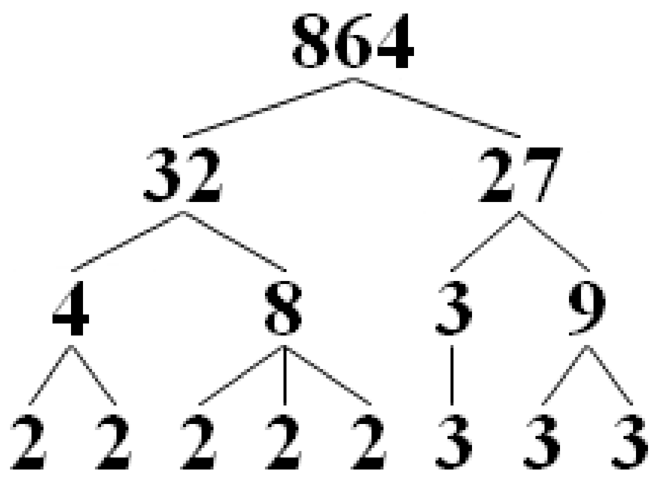

Functional (FE)
2013
Private keys can give different views of data.
Attribute-Based (ABE)
2005
Embeds complex access controls.
Identity-Based (IBE)
2001
Public key can be identity string, e.g., email address.
Public Key
1978
e.g., RSA
No need for a shared secret.

Prime factorization problem

Factorize number

$N = 864$

$= 2^5 \times 3^3$



Input: n-bits composite number N

Output: $N = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}, \alpha_k \in \mathbb{N}^*$

No classical algorithm has been published that can factor all integers in polynomial time.

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

Prime factorization problem

“Prime factorization one-way function!”

Input: large prime number p,q and a large number e

Easy to compute $\longrightarrow$

$$\begin{cases} n = p.q \\ d = e^{-1} \bmod (p-1)(q-1), \text{ e.d} = 1 \text{ mode } (p\text{-}1)(q\text{-}1) \\ C = M^{e} \bmod n \end{cases}$$

Input: n, e, C

$$\begin{cases} n = p.q \leftarrow p,q \\ d = e^{-1} \bmod (p-1)(q-1) \end{cases}$$

“Hard” to compute $\longleftarrow$

$$\boxed{C^{d} \bmod n = M^{e.d} \bmod n = M^{e.d \bmod (p-1)(q-1)} \bmod n = M}$$

The RSA Algorithm

Key Generation by Alice

Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calcuate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

$e.d = 1\ mod\ \phi(n)$

Encryption by Bob with Alice's Public Key

Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption by Alice with Alice's Public Key

Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

$$C^d mod\ n = (M^e)^d mod\ n$$
$$= M^{ed}\ mod\ n =?\ M$$

RSA Algorithm

- RSA makes use of an expression with exponentials
- Plaintext is encrypted in blocks with each block having a binary value less than some number *n*
- Encryption and decryption are of the following form, for some plaintext block *M* and ciphertext block C

$$C = M^e \bmod n$$

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

- Both sender and receiver must know the value of *n*
- The sender knows the value of *e,* and only the receiver knows the value of *d*
- This is a public-key encryption algorithm with a public key of *PU={e,n}* and a private key of *PR={d,n}*

Algorithm Requirements

- For this algorithm to be satisfactory for public-key encryption, the following requirements must be met:
  1. It is possible to find values of *e, d, n* such that $M^{ed} \bmod n = M$ for all $M < n$
  2. It is relatively easy to calculate $M^e \bmod n$ and $C^d \bmod n$ for all values of $M < n$
  3. It is infeasible to determine *d* given *e* and *n*

Exponentiation in Modular Arithmetic

- Both encryption and decryption in RSA involve raising an integer to an integer power, mod n
- Can make use of a property of modular arithmetic:

$$[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$$

- With RSA you are dealing with potentially large exponents so efficiency of exponentiation is a consideration

Algorithm for Computing a^b mod n

Note: The integer b is expressed as a binary number $b = b_k b_{k-1} \dots b_0$

$$a^b = a^{(b_k b_{k-1} \dots b_0)}$$

$$= a^{(2^k b_k + \dots + 2^2.b_2 + 2.b_1 + b_0)}$$

$$= \prod_{i=0}^{k} a^{b_i.2^i} = \prod_{i=0}^{k} (\underbrace{a^{b_i} . a^{.2^i}})$$

$c = 2^i$

$f_i = a^{.2^i}$

$f_{i+1} = a^{.2^{i+1}} = a^{2.2^i}$

$= (a^{2^i})^2 = (f_i)^2$

$f_i = a^c$

$c = 2^i + 1$

$f_{i+1} = (f_i)^2 . a$

```
c ← 0; f ← 1
for i ← k downto 0
    do  c ← 2 × c
        f ← (f × f) mod n
    if  b_i = 1
        then c ← c + 1
             f ← (f × a) mod n
return f
```

$c = 2^i$

$c = 2^i + 1$

Efficient Operation Using the Public Key

- To speed up the operation of the RSA algorithm using the public key, a specific choice of *e* is usually made
- The most common choice is 65537 ($2^{16} + 1$)
  - Two other popular choices are *e*=3 and *e*=17
  - Each of these choices has only two 1 bits, so the number of multiplications required to perform exponentiation is minimized
  - With a very small public key, such as *e* = 3, RSA becomes vulnerable to a simple attack

Efficient Operation Using the Private Key

- Decryption uses exponentiation to power *d*
  - A small value of *d* is vulnerable to a brute-force attack and to other forms of cryptanalysis
- Can use the Chinese Remainder Theorem (CRT) to speed up computation
  - The quantities *d* mod ($p - 1$) and *d* mod ($q - 1$) can be precalculated
  - End result is that the calculation is approximately four times as fast as evaluating $M = C^d \bmod n$ directly

Key Generation

- Before the application of the public-key cryptosystem each participant must generate a pair of keys:
  - Determine two prime numbers *p* and *q*
  - Select either *e* or *d* and calculate the other
- Because the value of *n* = *pq* will be known to any potential adversary, primes must be chosen from a sufficiently large set
  - The method used for finding large primes must be reasonably efficient

Procedure for Picking a Prime Number

- Pick an odd integer *n* at random
- Pick an integer *a* < *n* at random
- Perform the probabilistic primality test with a as a parameter. If *n* fails the test, reject the value *n* and go to step 1
- If *n* has passed a sufficient number of tests, accept *n;* otherwise, go to step 2

RSA: Confidentiality

A

$d_A \,/\, n_A, e_A$

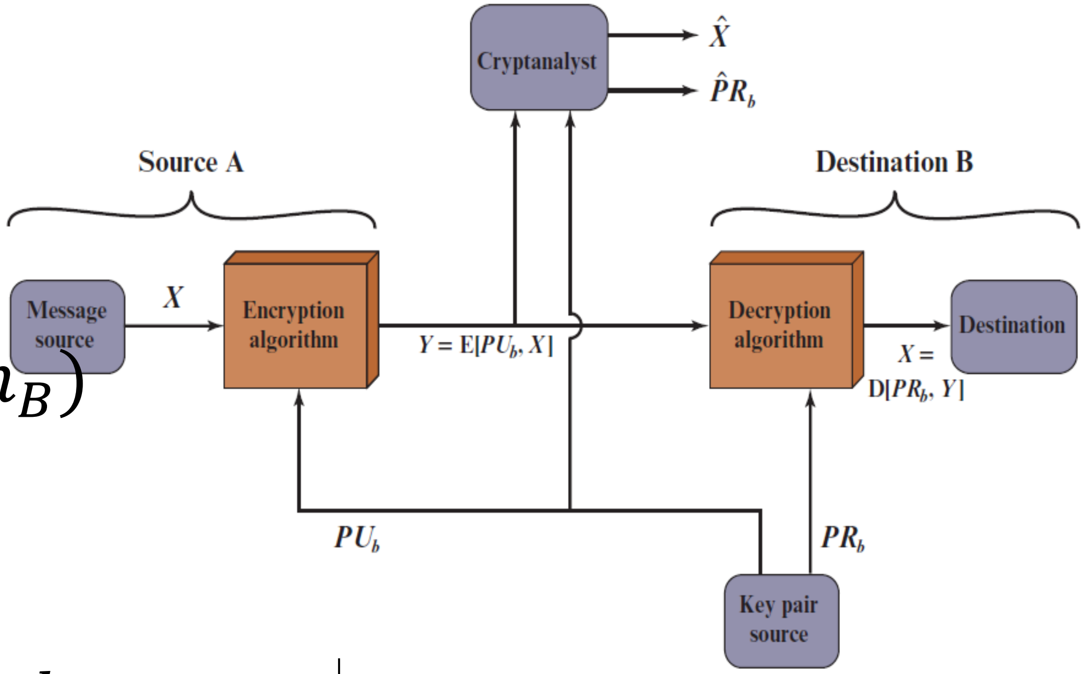
$e_A . d_A = 1 \bmod \phi(n_A)$

B

$d_B \,/\, n_B, e_B$

$e_B . d_B = 1 \bmod \phi(n_B)$

$$C = m^{e_B} \bmod n_B \longrightarrow$$



$$C^{d_B} \bmod n_B = (m^{e_B})^{d_B} \bmod n_B$$
$$= m$$

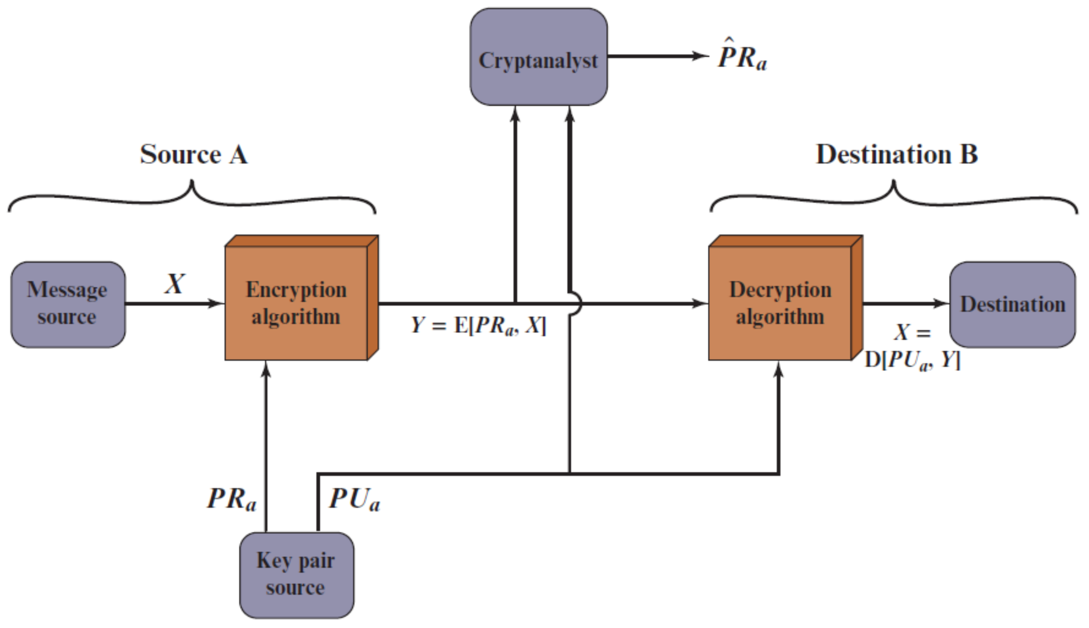
- **Protect secret key?**
- **Distribute public keys?**

RSA: Authentication

A **B**

d_A / n_A, e_A d_B / n_B, e_B

$S = m^{d_A} \bmod n_A$ (m, S)



Verify message ***m***

$$S^{e_A} \bmod n_A = (m^{d_A})^{e_A} \bmod n_A$$

$$= m'$$

$$? = m$$

- **Sent by A**
- **Original (integrity)**

- **Protect secret key?**
- **Distribute public keys?**

RSA: Authentication and Secrecy

A

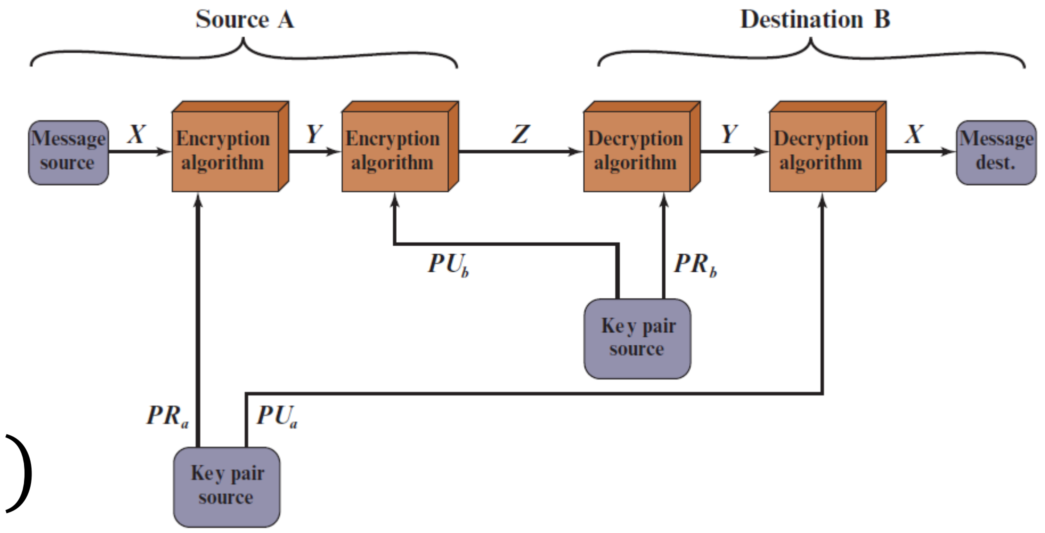
d_A / n_A, e_A

$S = k^{d_A} \bmod n_A$

B

d_B / n_B, e_B

$$\underrightarrow{(C_1 = k^{e_B}, S_1 = S^{e_B})}$$

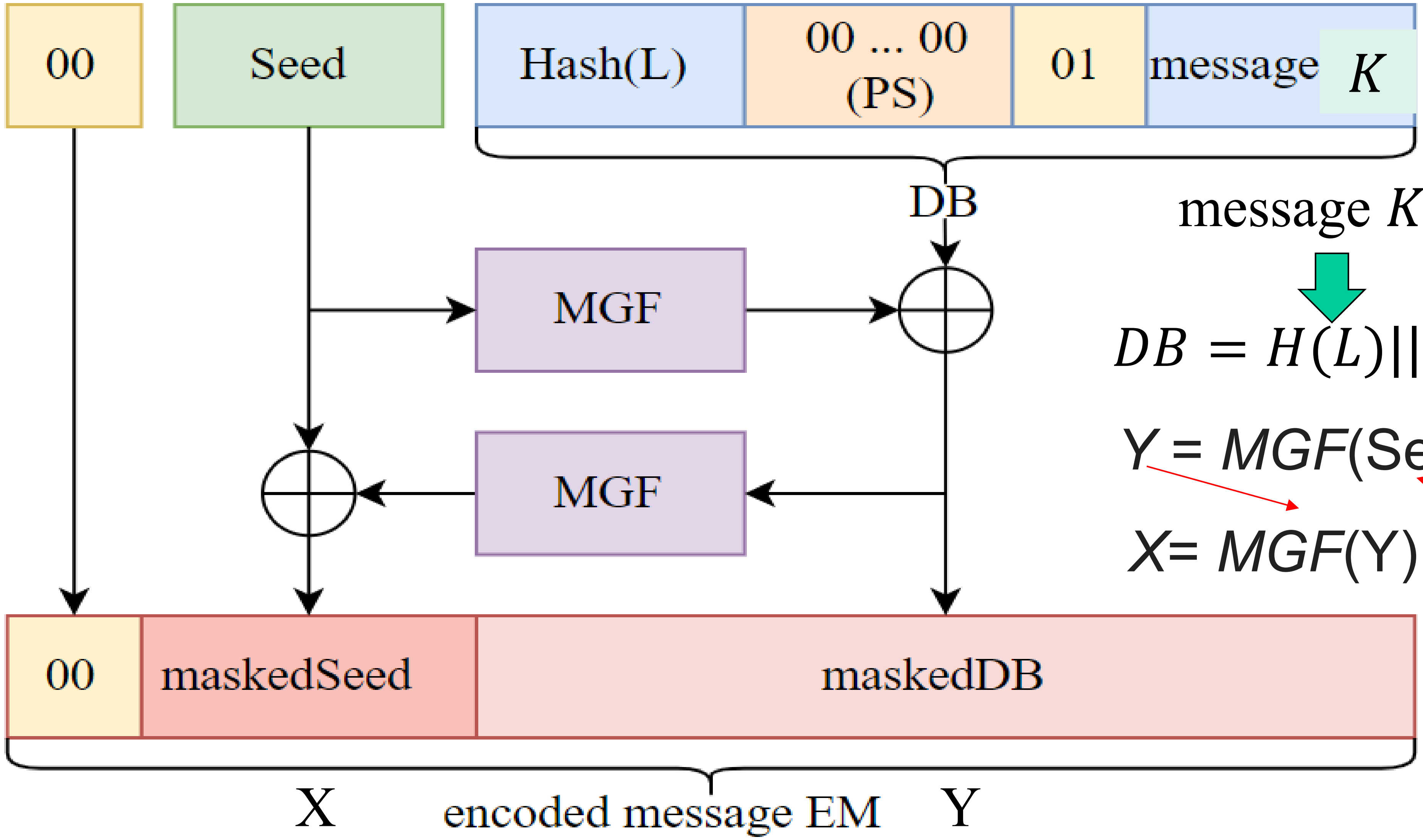


Limitation? ⇦

Decrypt and verify the secret key k

$$\left\{\begin{array}{l} C_1^{d_B} \bmod n_B = (k^{e_B})^{n_B} \bmod n_B = k; \\ S_1^{d_B} \bmod n_B = (S^{e_B})^{n_B} \bmod n_B = S; \\ S^{e_A} \bmod n_A = (k^{d_A})^{e_A} \bmod n_A = k' =?\, k \end{array}\right.$$

Encryption Using Optimal Asymmetric Encryption Padding (OAEP)



MGF: a hash function

message K

$$DB = H(L)||0000..||01||\underline{\underline{K}}$$

$$Y = MGF(\text{Seed}) \oplus \underline{\underline{DB}}$$

$$X = MGF(Y) \oplus \underline{\underline{\text{Seed}}}$$

?

Outline

- Why asymmetric cryptography?
- Factoring Based Cryptography (P1)
- Logarithm Based Cryptography (P2)
  - ElGamal cipher;
  - Diffie-Hellman key exchange;
- Elliptic Curve Cryptography (P3)
- Some advanced cryptography system (quantum resistance)

Discrete Logarithm problem

Finitemultiplicative group $(G,.) =< g > = \{g^n : n \in \mathbb{Z}\}$

Example: $G = \mathbb{Z}_p \backslash \{0\} = \{1,2,\dots,p-1\} =< g >$

$$g, n \xrightarrow{\text{Easy to compute}} y_0 = g^n \bmod p$$

$$g^n = y_0 \bmod p \xleftarrow{\text{Hard to solve } n} g, y_0, p$$

```
AutoSeededRandomPool prng;
Integer p, q, g;
CryptoPP::PrimeAndGenerator pg;

pg.Generate(1, prng, 512, 511);
p = pg.Prime();
q = pg.SubPrime();
g = pg.Generator();
```

Hard to solve equation $g^x = a \bmod p$ in finite field!

Outline

- Why asymmetric cryptography?
- Factoring Based Cryptography (P1)
  - RSA signature;
- Logarithm Based Cryptography (P2)
  - ElGamal cipher
  - Diffie-Hellman key exchange;
- Elliptic Curve Cryptography (P3)
- Some advanced cryptography system (quantum resistance)

ElGamal cipher

ElGamal parameters

Large prime number: p

Multiplicative group

$$G = \langle g \rangle = \mathbb{Z}_p \backslash \{0\} = \{1,2,\dots,p-1\}$$

Key generation (

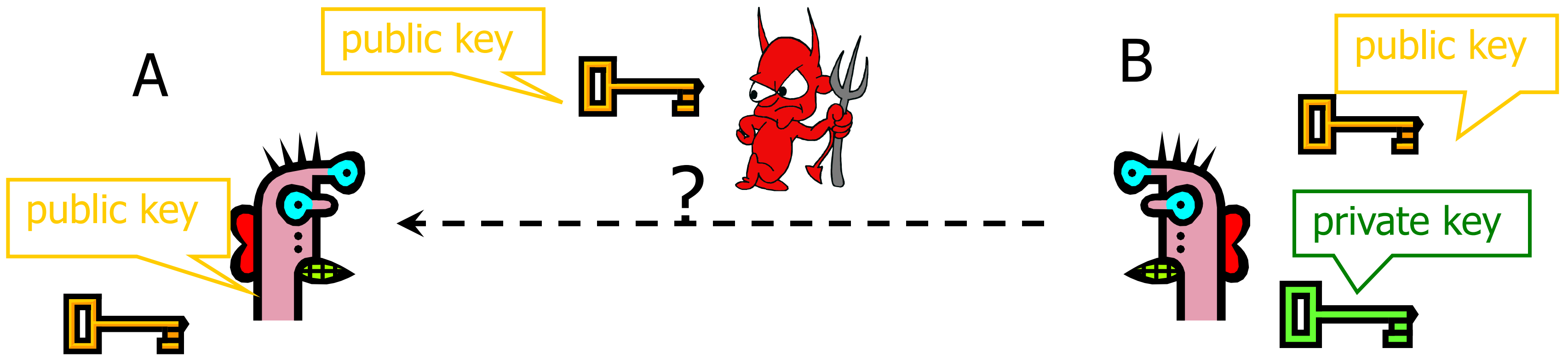
Secret key: $x \in_R [1, p-1]$

Public key: $h = g^x \bmod p \in \mathbb{Z}_p$

ElGamal cipher

- **Encryption message $m < p - 1$ (using public key $h = g^{x}$)**
  - Choose a random number: $r \in_R [1, p-1]$
  - Compute $C_1 = g^r \bmod p$;
  - Computer $C_2 = m.h^r \ mod\ p$
  - Output cipher message (C_1, C_2)
- **Decryption (C_1, C_2) (using secret key x)**
  - Compute $(C_1)^x mod\ p = g^{r.x} \bmod p$;
  - Computer $\dfrac{C_2}{(C_1)^x}\ mod\ p = \dfrac{m.g^{x.r}}{g^{r.x}}\ mod\ p$
    $= m$
  - Output message m

ElGamal cipher



$PU_B = h_B, p$

$(PR_B = x_B, PU_B = h_B = g^{x_B}, p)$

Input: $M < p$

Select a random number: $r < p - 1$

Compute: $C_1 = g^r \bmod p$

$C_2 = m.h_B^r \bmod p$

$\xrightarrow{(C_1, C_2)}$

Compute:

$$\frac{C_2}{(C_1)^{x_B}} \bmod p$$

$$= \frac{m.g^{x_B.r}}{g^{r.x_B}} \bmod p = m$$

Outline

- Why asymmetric cryptography?
- Factoring Based Cryptography (P1)
  - RSA signature;
- Logarithm Based Cryptography (P2)
  - ElGamal cipher
  - Diffie-Hellman key exchange;
- Elliptic Curve Cryptography (P3)
- Some advanced cryptography system (quantum resistance)

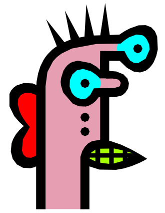
Diffie-Hellman key exchange

- A and B never met and share no secrets;
- Public info: the prime number p and g
  - ➢ p is a large prime number, g is a generator of Z_p^*
    - • $Z_p^* = \{1, 2 \ldots p\text{-}1: \forall a \in Z_p^*\ \exists i$ such that $a = g^i \bmod p\}$

Pick secret, random x

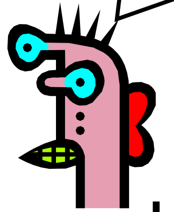
Pick secret, random y

A



$$X = g^x \bmod p$$

$$Y = g^y \bmod p$$



B

Compute

$$k_A = Y^x \bmod p = (g^y)^x \bmod p$$
$$= g^{y.x} \bmod p$$

Compute

$$k_B = X^y = (g^x)^y \bmod p$$
$$= g^{xy} \bmod p$$

Session key $K = k_A = k_B = g^{x.y}$ ➡ Symmetric key

Diffie-Hellman exchange Protocol (DHE)

p = 16069380442589902755419620923411626025222029937827928353013O1

g = 123456789

g^a mod p =
7846737452942265357975459631985270257549969298008577794859 3

g^b mod p =
5600481042932181286674410213424831338026262713942994101287 98

$a =$
6854080036270637610592759196657816943686394595278718815314 52

$b =$
3620591319129419876378802573252696966828367355249422468074 40

$(g^b)^a \; mod \; p$

g^{ab} mod p =
43745285708580178521996144300O
845969831329749878767465041215

$(g^a)^b mod \; p$

Why Is Diffie-Hellman Secure?

- Discrete Logarithm (DL) problem:
  given g^x mod p, it's hard to extract x
  - There is no known efficient algorithm for doing this
  - This is not enough for Diffie-Hellman to be secure!
- Computational Diffie-Hellman (CDH) problem:
  given g^x and g^y, it's hard to compute g^{xy} mod p
  - … unless you know x or y, in which case it's easy
- Decisional Diffie-Hellman (DDH) problem:
  given g^x and g^y, it's hard to tell the difference between g^{xy} mod p and g^r mod p where r is random

Properties of Diffie-Hellman

- Assuming DDH problem is hard, Diffie-Hellman protocol is a secure key establishment protocol against passive attackers
  - Eavesdropper can’t tell the difference between the established key and a random value
  - Can use the new key for symmetric cryptography
- Basic Diffie-Hellman protocol does not provide authentication
  - IPsec combines Diffie-Hellman with signatures, anti-DoS cookies, etc.

Man-in-the middle attacks the DHE

$$\begin{tikzcd}(a, g^a) \arrow[r, shift left, "g^a \bmod p"] & (b, g^b) \arrow[l, shift left, "g^b \bmod p"]\end{tikzcd}$$

A(Putin)

B()

$$sk = (g^b)^a = (g^a)^b = g^{ab} mod\ p$$

man-in-the-middle attack!

$$\begin{tikzcd}{} \arrow[r, shift left, "g^a \bmod p"] & {} \arrow[l, shift left, "g^m \bmod p"] \arrow[r, shift left, "g^m \bmod p"] & {} \arrow[l, shift left, "g^b \bmod p"]\end{tikzcd}$$

$$sk_1 = g^{am} mod\ p$$

$$sk_2 = g^{bm} mod\ p$$

Advantages of Pblic-Key Crypto

- Confidentiality without shared secrets
  - Very useful in open environments
  - Can use this for key establishment, avoiding the “chicken-or-egg” problem
    - With symmetric crypto, two parties must share a secret before they can exchange secret messages
- Authentication without shared secrets
- Encryption keys are public, but must be sure that Alice’s public key is really <u>her</u> public key
  - This is a hard problem… Often solved using public-key certificates

Disadvantages of Public-Key Crypto

- Calculations are 2-3 orders of magnitude slower
  - Modular exponentiation is an expensive computation
  - Typical usage: use public-key cryptography to establish a shared secret, then switch to symmetric crypto
    - SSL, IPsec, most other systems based on public crypto
- Keys are longer
  - 3072 bits (RSA) rather than 128 bits (AES)
- Relies on unproven number-theoretic assumptions
  - Factoring, RSA problem, discrete logarithm problem, decisional Diffie-Hellman problem…