# Asymmetric Crypto

# Current Cryptographic Standards

## Encryption
(Privacy & Confidentiality)

## Digital Signatures
Authentication / Non Repudiation
Message Integrity

### Symmetric Key Algorithm

**Stream Cipher**
RC4
SEAL
WAKE
A5
PKZIP

**Block Cipher**
AES, DES, 3DES
RC2,RC5,RC6
Blowfish, Twofish, Serpent
Mars, Cast, Idea

### Public Key
Asymmetric Algorithm

**Discrete Log**
DSA
ECC
**Diffe-Helman**
ElGamal

**Factoring**
**RSA**
LUC

### Message Digest
Hash Algorithm
MD2
MD5
SHA
SHA-1
SHA-2
SHA-3
Ripe-MD160

**Key Management**
ISA/KMP
IKE
SKIP
Photuris
Diffe-Helman
ElGamal

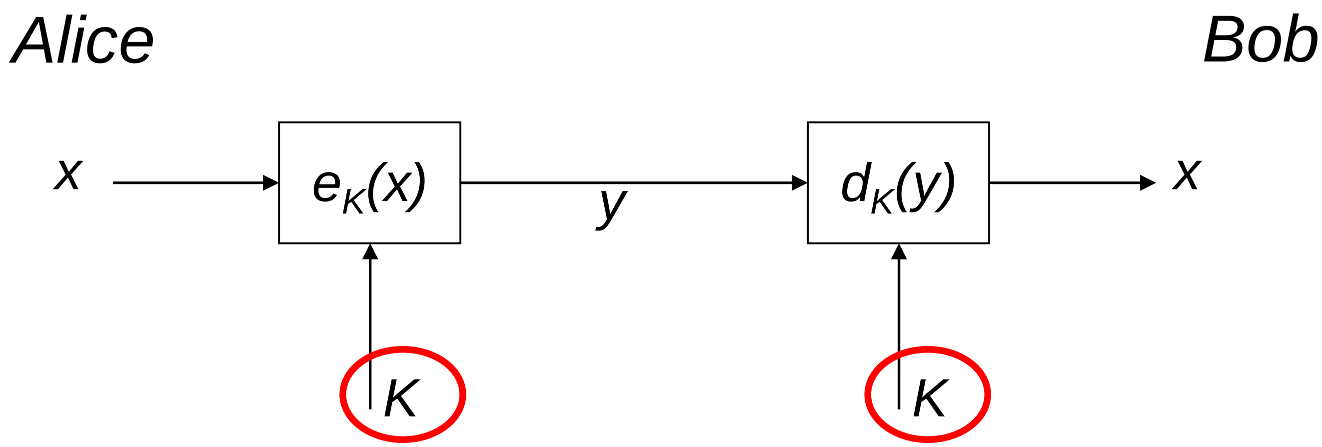
**Public Key Infrastructure (PKI)**
PXIX
SPKI
SDS
PGP
DNSEC

$$c = e_{PuK}(m)$$

$$m = d_{PrK}(c)$$

$$e_{PrK}(m)$$

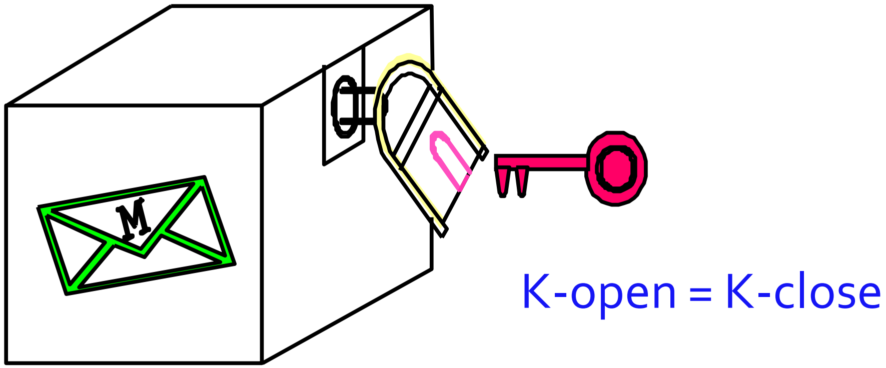
# Cryptography revisited




Two properties of symmetric (secret-key) crypto-systems:

- The **same secret key *K*** is used for encryption and decryption
- Encryption and Decryption are very similar (or even identical) functions

# Symmetric Cryptography: Analogy




Safe with a strong lock, only Alice and Bob have a copy of the key

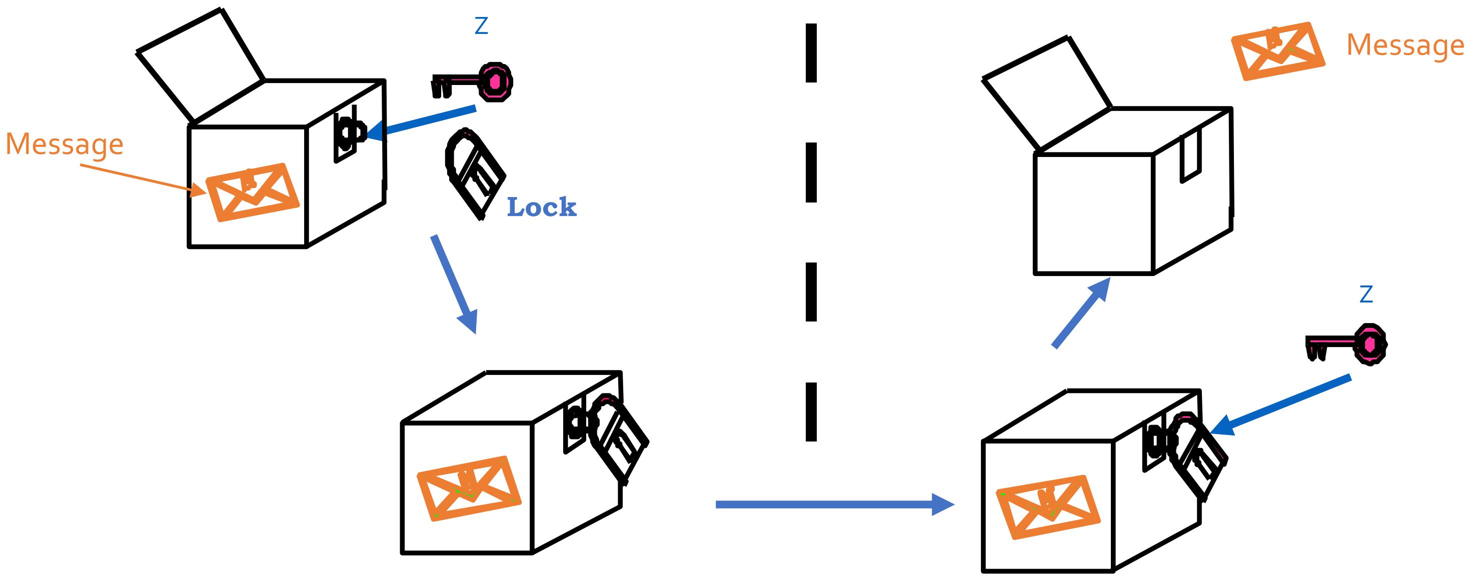
- Alice encrypts → locks message in the safe with her key
- Bob decrypts → uses his copy of the key to open the safe

- Open and close using shared secret keys
- Secret key agreement required !

# Symmetric Cryptography: Analogy

*SENDER* *RECEIVER*

**Key = Z ⟷ Secret key agreement ⟷ Key = Z**



# Symmetric Cryptography Shortcomings

- Key distribution problem: The secret key must be **transported securely**
- Number of keys: In a network, each pair of users requires an individual key

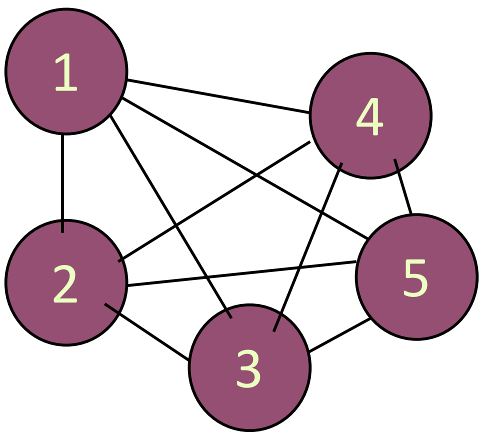
→ *n* users in the network require $\frac{n \cdot (n-1)}{2}$ keys, each user stores *(n-1)* keys

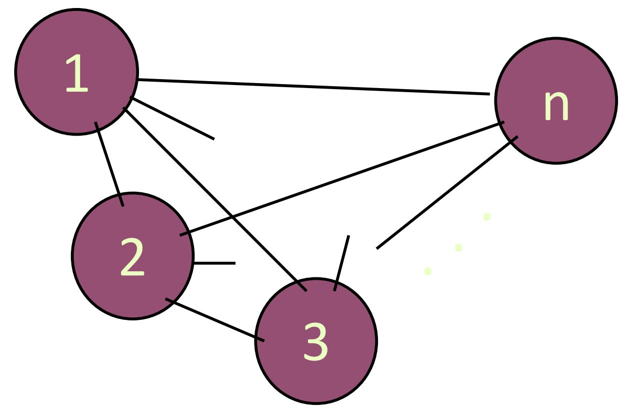
- No support for message **integrity**: verifying that a message comes intact from the sender
- No support for "**non-repudiation**"
  - **Example**: Alice can claim that she never ordered a TV on-line from Bob (he could have fabricated her order)


6

# Symmetric Cryptography Shortcomings

Question: How many secret-keys needed to be exchanged in order to set up a system of n-users?




10 key-exchanges for 5 users

$\frac{n\,(n-1)}{2}$ keys for n users

For 10 000 users we need 50 million key-exchanges !

Public-Key system **drops out** the secret key-agreement completely

# Idea behind Asymmetric Cryptography

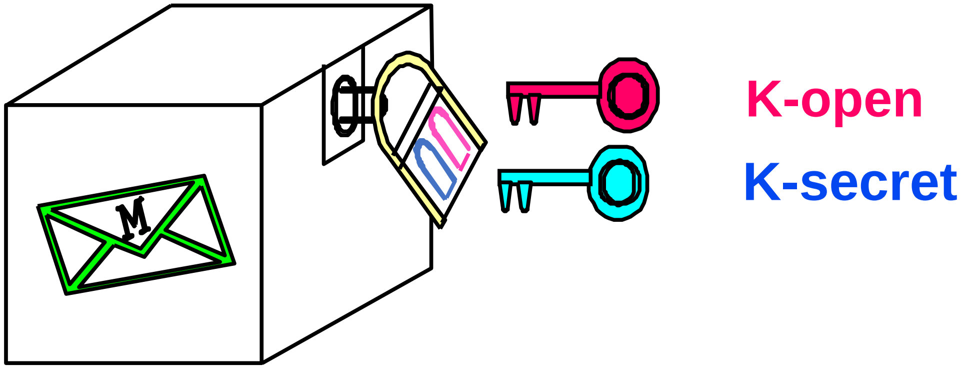
**New Idea:**

Use the “good old mailbox“ principle:

**Everyone** can drop a letter

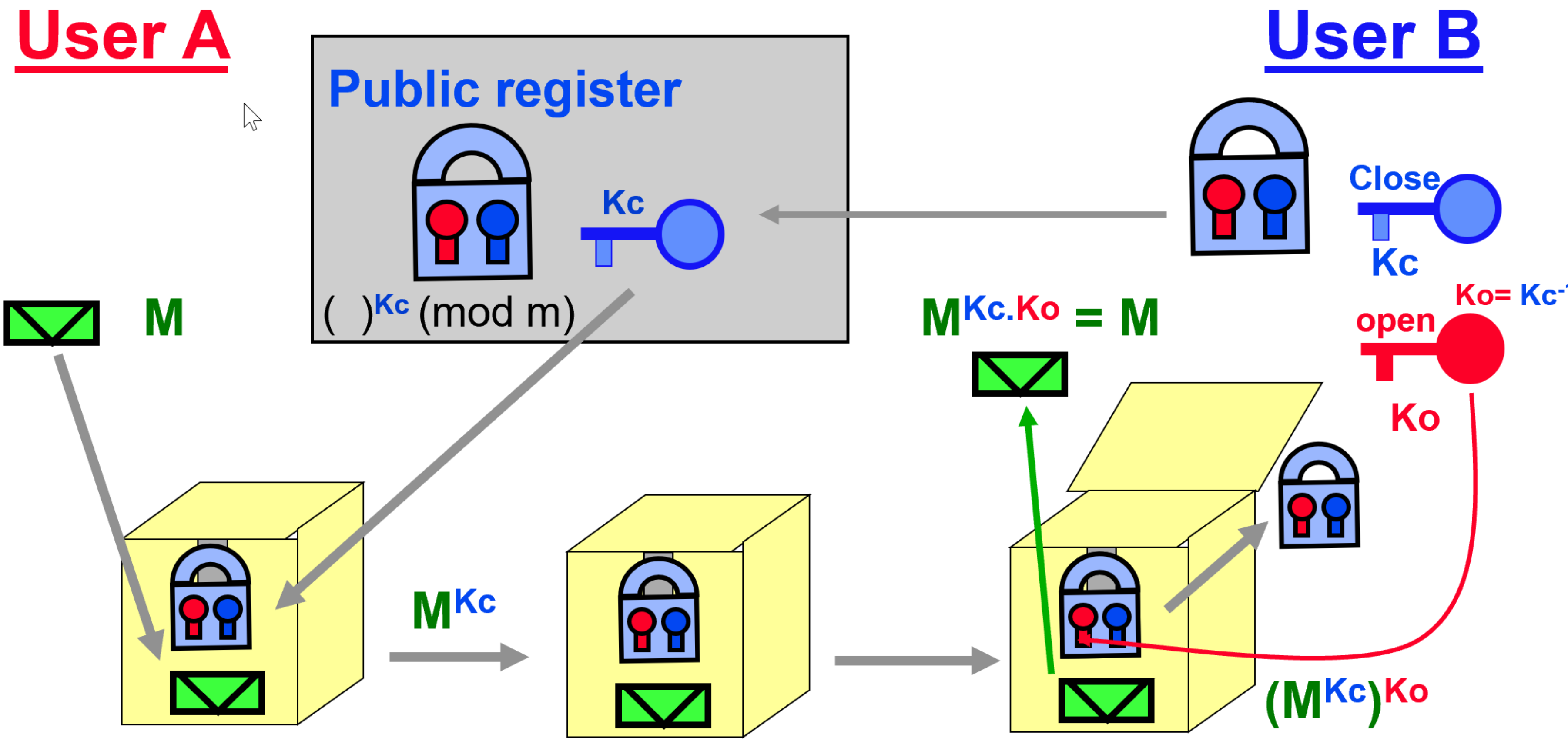
**But: Only the owner** has the correct key to open the box

# Asymmetric Cryptography: Analogy

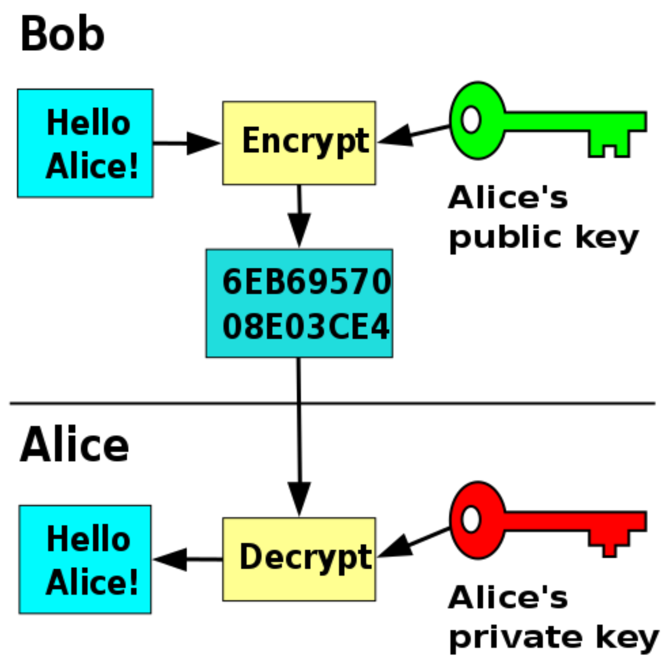



- Open and close with <u>different  keys</u>!!
- <u>No Secret</u> Key Agreement required

# Asymmetric Cryptography: Analogy User B gets a secured message from A)



# Asymmetric Cryptography




→ **Key Distribution Problem** solved

→ Anyone can encrypt messages using the public key, but only the holder of the **paired private key** can decrypt. Security depends on the secrecy of the private key.

→ During the key generation, a key pair $K_{pub}$ and $K_{pr}$ is computed

# Public Key Cryptography Applications

The asymmetric cryptography (e.g., RSA) are mainly used for:

- **Key Distribution** without a pre-shared secret (key)
- **Digital Signatures** to provide message integrity and non-repudiation

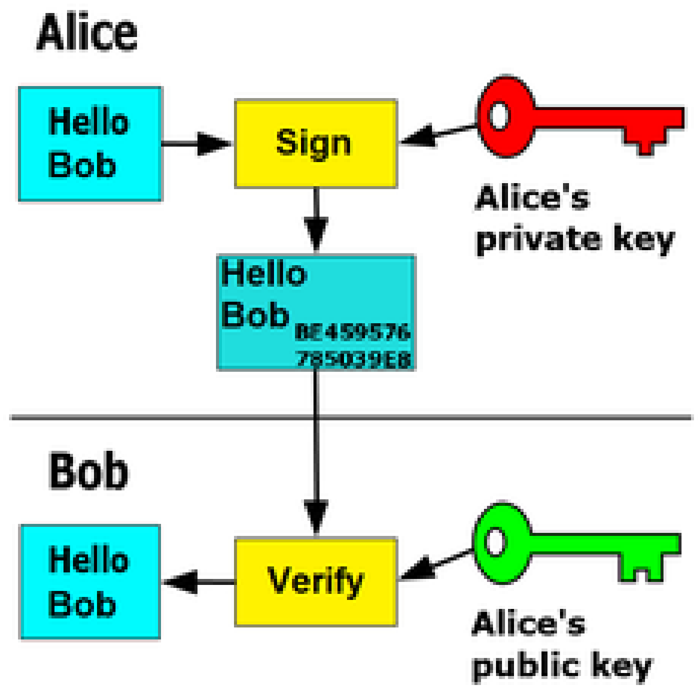
Rarely used for:

- **Encryption** because it is computationally very intensive (1000 times slower than symmetric Algorithms!)

# Applications for Public-Key Cryptosystems

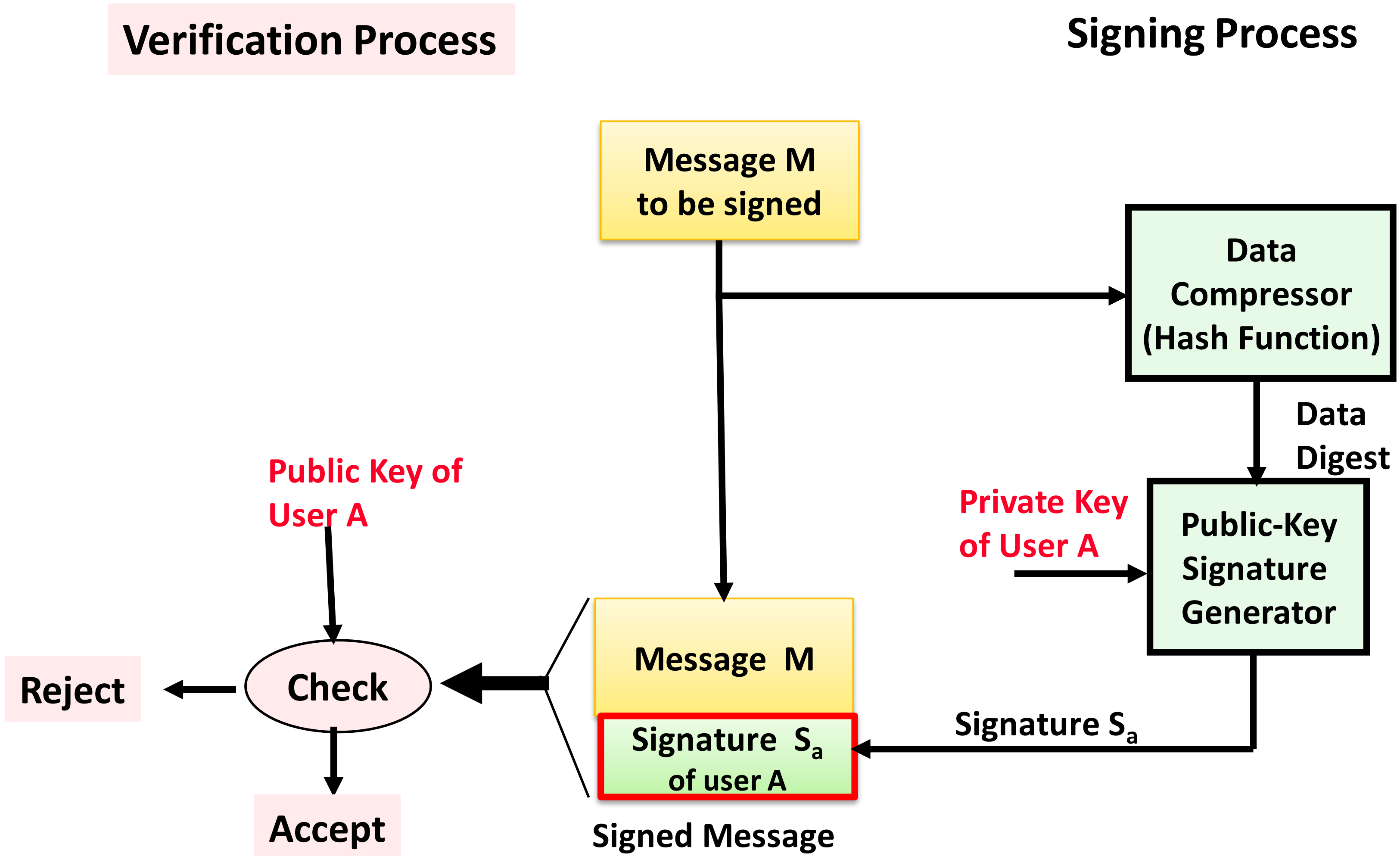
| Algorithm | Encryption/Decryption | Digital Signature | Key Exchange |
|---|---|---|---|
| RSA | Yes | Yes | Yes |
| Elliptic Curve | Yes | Yes | Yes |
| Diffie-Hellman | No | No | Yes |

# Digital Signatures




- 1) Alice signs a message with her private key
- 2) Bob can verify that Alice sent the message (i.e., non-repudiation) and that the message has not been modified (i.e., integrity)

# Public-Key Signature Scheme



# Basic Key Transport Protocol 1/2

In practice: **Hybrid systems**, incorporating asymmetric and symmetric algorithms

1. **Key exchange** (for symmetric schemes) and **digital signatures** are performed with (slow) **asymmetric** algorithms

2. **Encryption** of data is done using (fast) symmetric ciphers, e.g., **block ciphers or stream ciphers**

# Basic Key Transport Protocol 2/2

Example: Hybrid protocol with AES as the symmetric cipher

| Alice | | Bob |
|---|---|---|
| | | $(K_{pubB}, K_{prB}) = K$ |
| | $\xleftarrow{K_{pubB}}$ | |
| Choose random symmetric key $K$ | | |
| $y_1 = e_{K_{pubB}}(K)$ | $\xrightarrow{y_1}$ | |
| | | $K = d_{K_{prB}}(y_1)$ |
| message x | | |
| $y_2 = AES_K(x)$ | $\xrightarrow{y_2}$ | $x = AES^{-1}_K(y_2)$ |

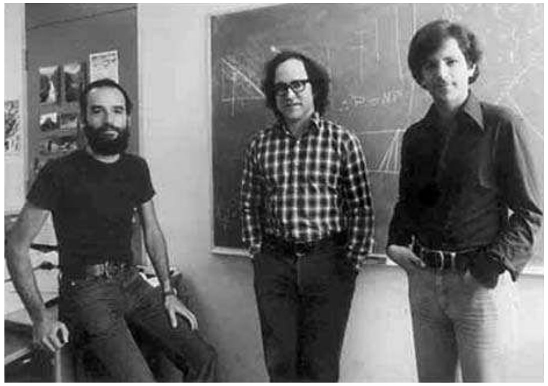
**Key Exchange (asymmetric)**

**Data Encryption (symmetric)**

# Key Exchange with Public Key Crypto

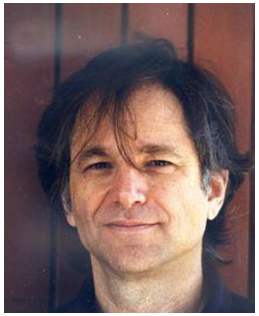
- Alice creates a secret key, encrypts it with Bob's public key and sends it off
- Bob decrypts the message with his private key
- Use shared key for further communication
- This is how many applications work
- Could encrypt/decrypt using public key cryptography but it is slow

# RSA Cryptosystem



**Rivest**

**Shamir**



**Adleman**

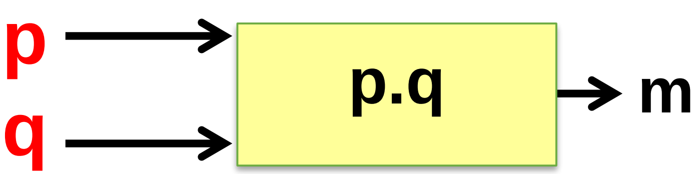
- Developed by Rivest, Shamir, and Adleman in 1977
- Until now, RSA is the most widely use asymmetric cryptosystem although elliptic curve cryptography (ECC) becomes increasingly popular
- RSA is mainly used for two applications: Key exchange & Digital signatures

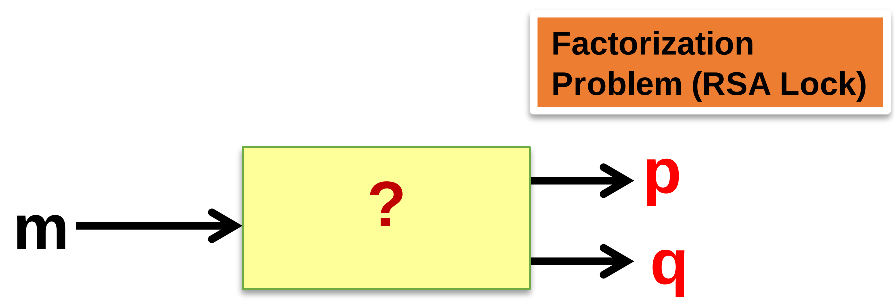
# RSA One-way Function (Lock)

Asymmetric schemes are based on a "**one-way function**" ***f()***:

- Computing *y = f(x)* is computationally easy
- Computing the inverse $x = f^{-1}(y)$ is computationally infeasible
- One way functions are based on **mathematically hard problems**
- RSA security relies on the difficulty of factoring large integers
  - Multiplying two primes is easy (e.g., 1889 x 3547 = 6,700,283 )
  - Given a composite integer *n*, find its prime factors is mathematically hard!

(e.g., It is hard to find $\text{Prime}_1$ and $\text{Prime}_2$ such that $\text{Prime}_1 \times \text{Prime}_2 = 6{,}700{,}283$ )



# Encryption and Decryption

- Encryption and decryption are simply exponentiations

**Definition**

Given the public key $(n,e) = k_{pub}$ and the private key $d = k_{pr}$ we write

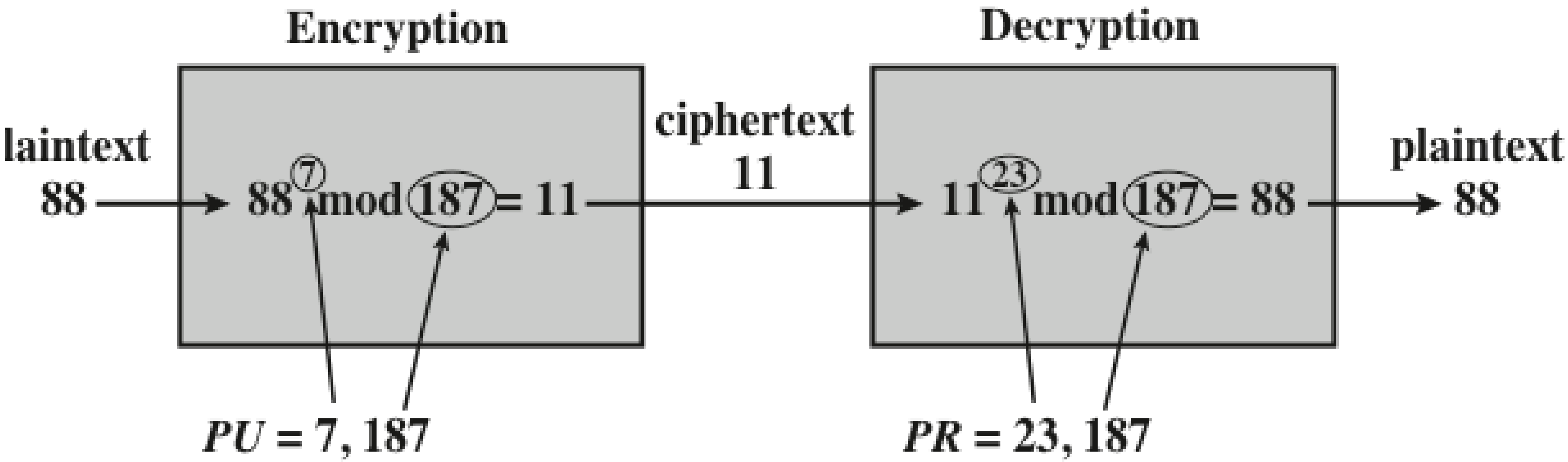
$$c = e_{k_{pub}}(m) = m^{e} \bmod n$$

$$m = d_{k_{pr}}(c) = c^{d} \bmod n$$

$e_{k_{pub}}()$ is the encryption operation and $d_{k_{pr}}()$is the decryption operation.

- In practice **d** and **n** are very long integer numbers (≥ 1024 bits)
- The security of the scheme relies on the fact that it is hard to derive the private key **d** given the public-key (n, e)

# RSA Example



Encryption
Decryption
laintext
88
88 7 mod 187 = 11
ciphertext
11
11 23 mod 187 = 88
plaintext
88
PU = 7, 187
PR = 23, 187

# Key Generation

- RSA has set-up phase during which the private and public keys are computed

**Algorithm: RSA Key Generation**

**Output**: public key: $k_{pub} = (n, e)$ and private key $k_{pr} = d$

1. Choose two large primes $p, q$
2. Compute $n = p * q$
3. Compute $\Phi(n) = (p-1) * (q-1)$
4. Select the public exponent $e$ such that $1 < e < \Phi(n)$ *and does not share any factor with* $\Phi(n)$
5. Compute the private key $d$ such that $(d * e) \bmod \Phi(n) = 1$

the extended **Euclidean algorithm** is used to compute $d$.

**6. RETURN** $k_{pub} = (n, e), k_{pr} = d$

Φ is called Phi

# Example: RSA with small numbers

**ALICE** | **BOB**

Message $\mathbf{m = 4}$

1. Choose $p$ = 3 and $q = 11$
2. Compute $n = p * q = 33$
3. $\Phi(n) = (3-1) * (11-1) = 20$
4. Choose $e = 3$
5. $d \equiv e^{-1} \equiv 7 \bmod 20 = 7$

$K_{pub}$ = (33,3) ←

$c = m^e = 4^3 \bmod 33 = 31$

c = 31 →

$c^d = 31^7 \bmod 33 = \mathbf{4}$

# Asymmetric Key Cryptography – RSA Encryption Algorithm

- Message: $m = 3$
- Choose 2 random, prime numbers: $p = 19$, $q = 13$
- $n = pq$, $n = 247$
- Choose a random # to be **e** (encryption key): $e = 7$
- $\Phi(n) = (p-1)(q-1) = 216$
- Compute **d** (decryption key) (private key)

  $d * e \bmod \Phi(n) = 1$ **(need to solve for d)**

  $d = 31$ (using Extended Euclidean Algorithm)
- Public key = $(n,e) = (247,7)$
- To encrypt: $c = m^e \bmod n$ ➔ $c = 3^7 \bmod 247$ ➔
  $c = 211$ (ciphertext)
- To decrypt: $m = c^d \bmod n$ ➔ $m = 211^{31}$ mode $247$ ➔
  $m = 3$ (plaintext)

# Attacks and Countermeasures

- **Brute force key search**
  - using exhaustive search for factoring of n in order to obtain $\Phi(n)$
  - Can be prevented using a sufficiently large modulus *n*
  - The current factoring record is 664 bits. Thus, it is recommended that *n* should have a bit length between 1024 and 3072 bits
- Implementation attacks such **Side-channel analysis**
  - Exploit physical leakage of RSA implementation (e.g., power consumption, etc.)
  - Timing attacks on running of decryption can Infer operand size based on time taken

# Summary

- RSA is the most widely used public-key cryptosystem
- RSA is mainly used for key transport and digital signatures
- RSA relies on the fact that it is hard to factorize *n*
- Currently 1024-bit cannot be factored, but progress in factorization could bring this into reach within 10-15 years. Hence, RSA with a 2048 or 3076 bit modulus should be used for long-term security

# References

- RSA Wikipedia page

https://en.wikipedia.org/wiki/Public-key_cryptography

https://en.wikipedia.org/wiki/RSA_(cryptosystem)