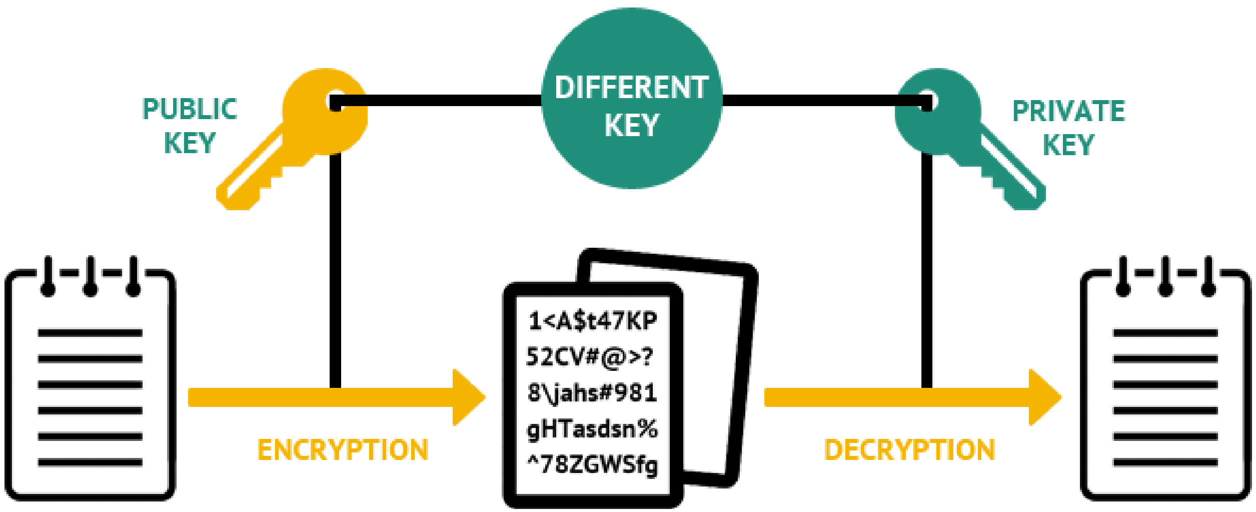
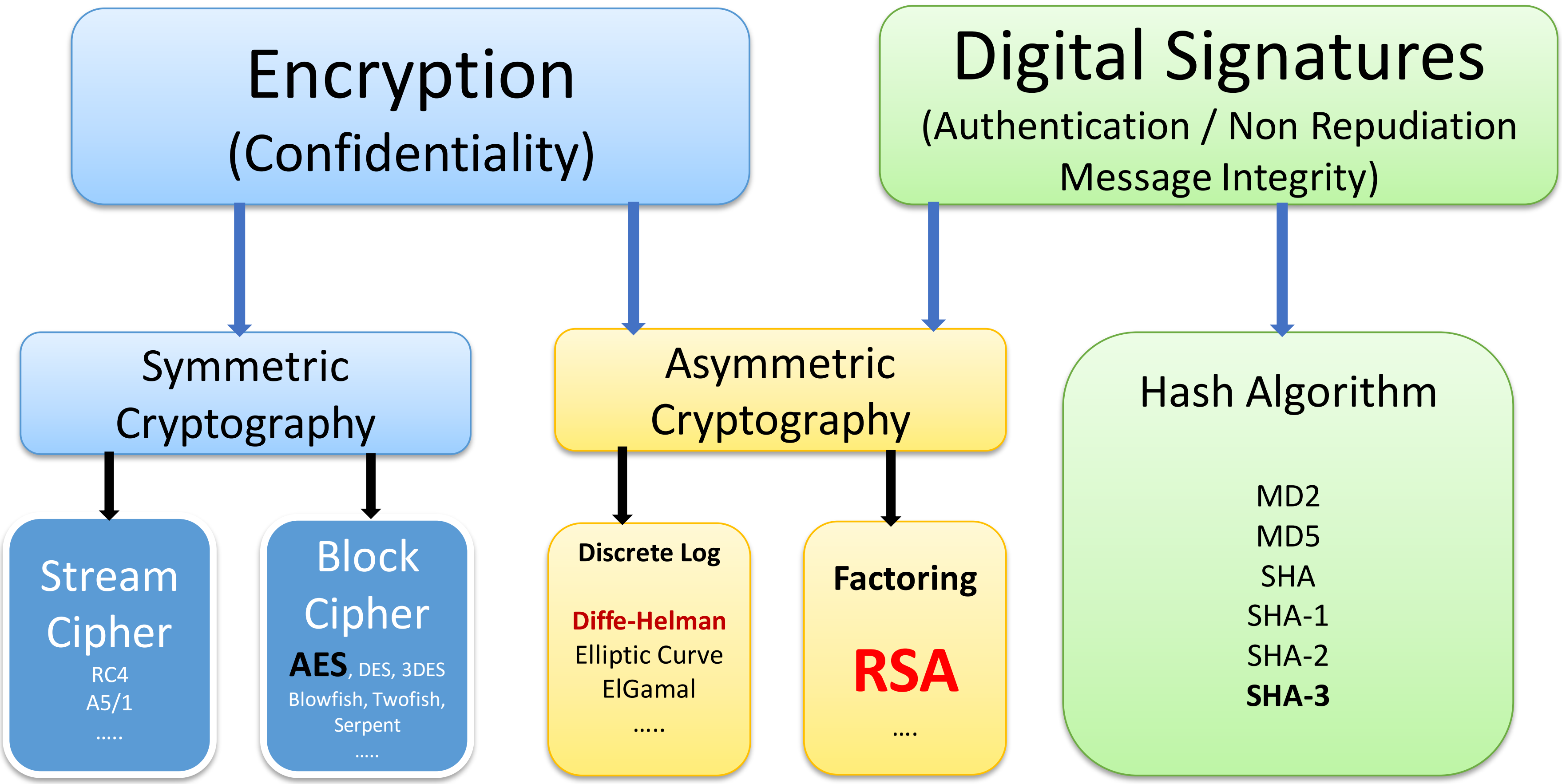
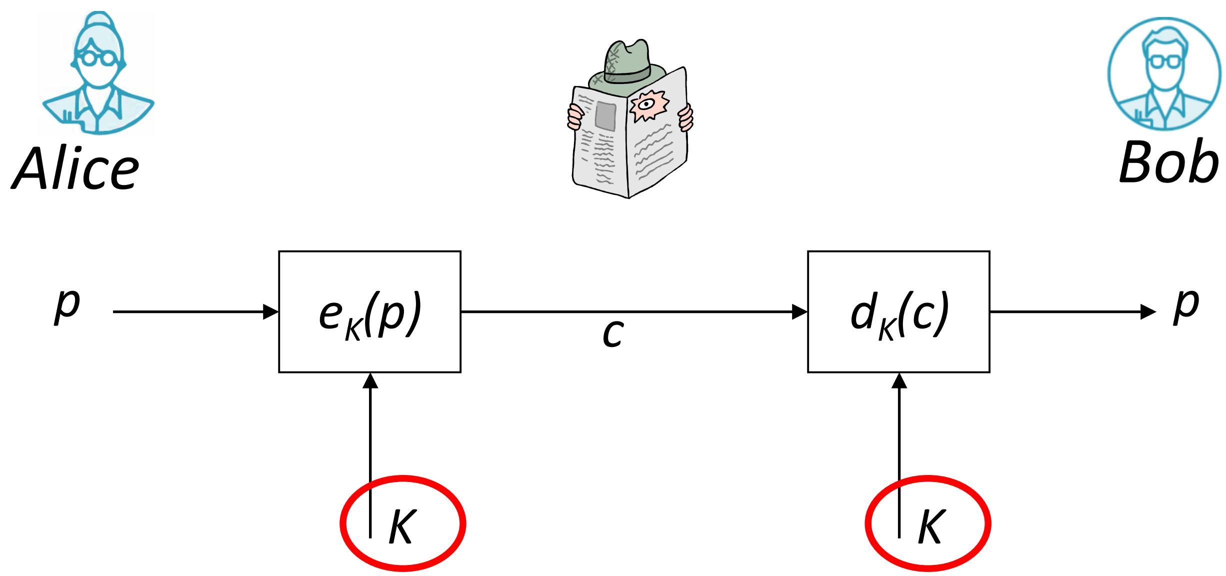
Asymmetric Cryptography



Current Cryptographic Standards



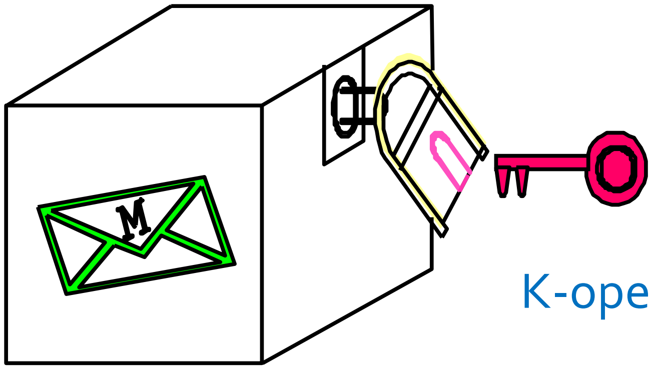
Cryptography revisited



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

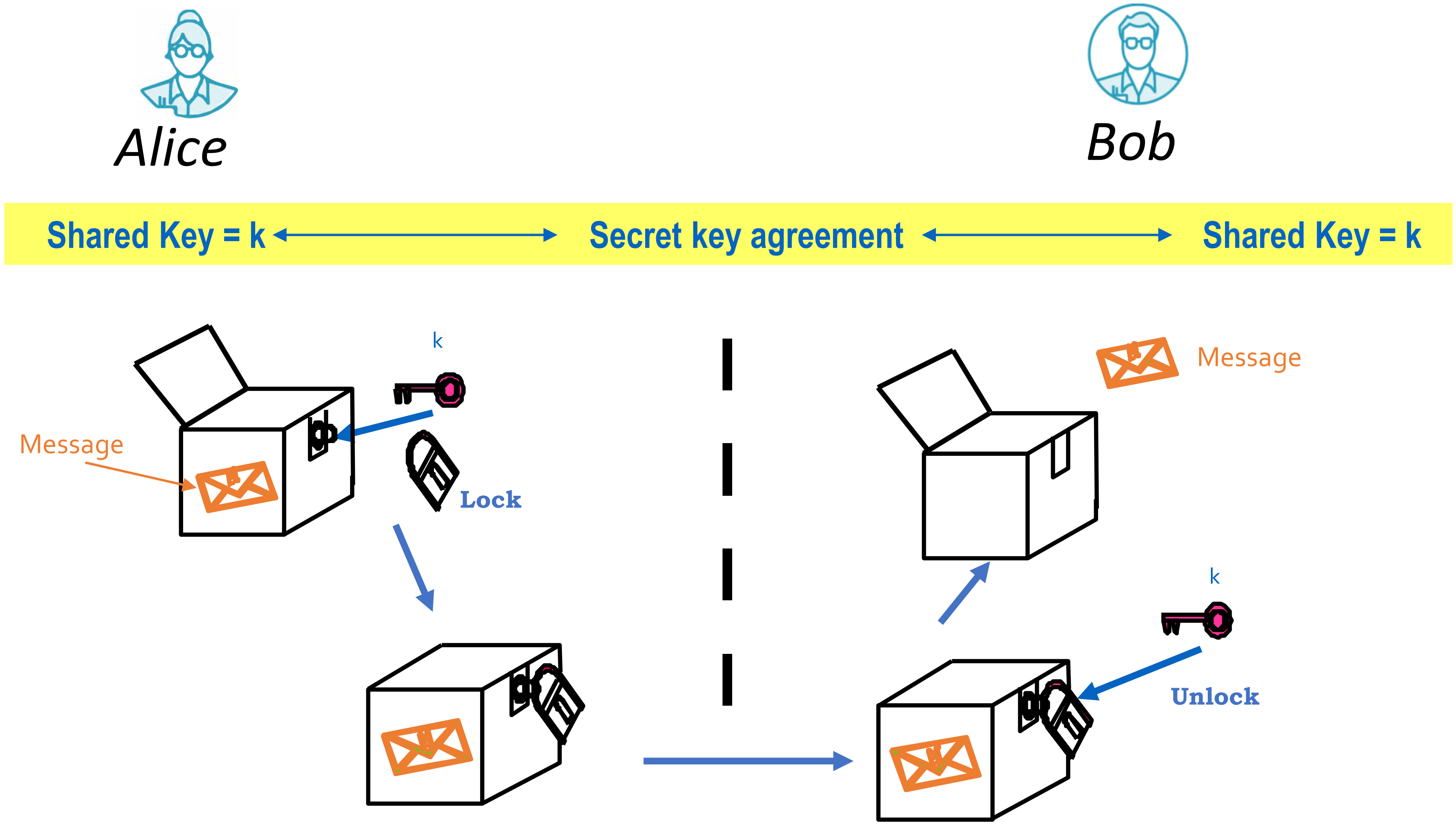


Safe with a strong lock, only Alice and Bob have a copy of the key

- 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



Symmetric Cryptography Shortcomings

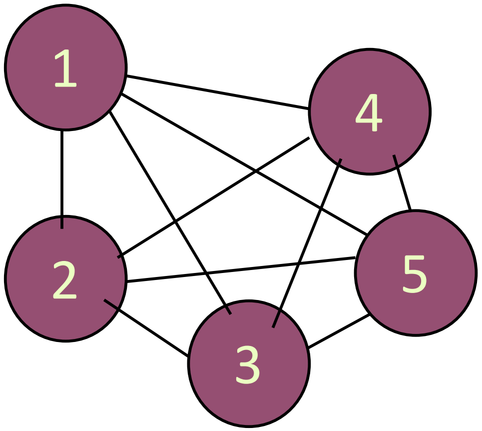
- Key distribution problem: The secret key must be **exchanged securely**
  - Establish a shared key with each entity we want to communicate with
- Number of keys: In a network, each pair of users requires an individual key

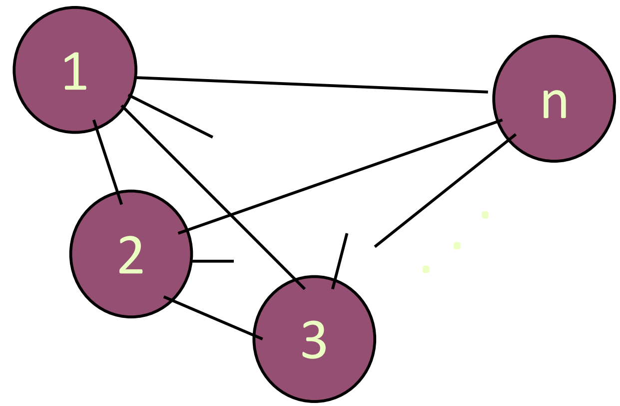
→ *n* users in the network require $\frac{n \cdot (n-1)}{2}$ keys, each user stores *(n-1)* keys

- 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)

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

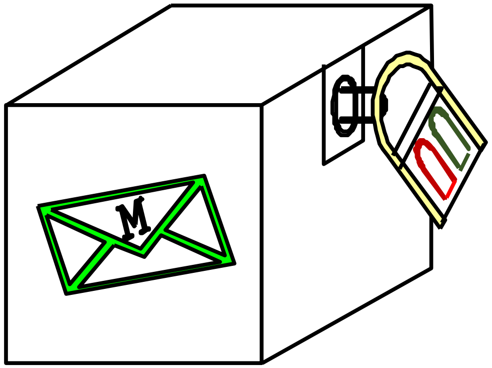
New Idea:

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

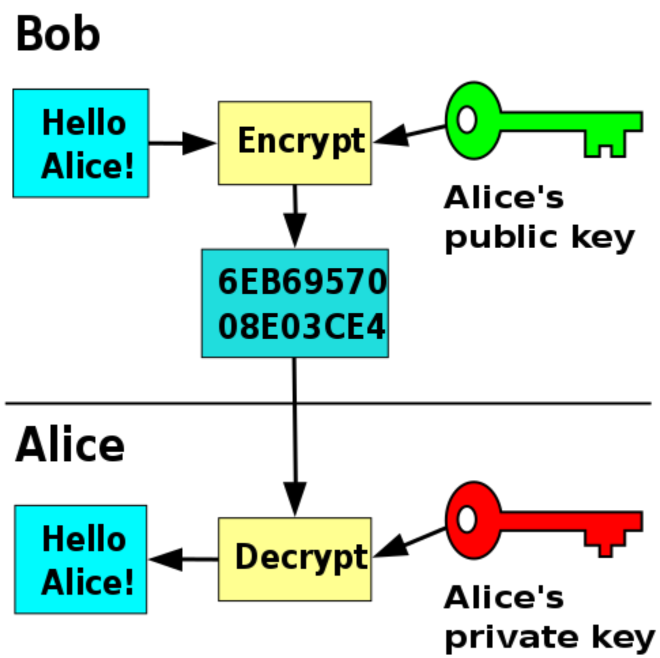


K-lock (public)

K-unlock (private)

- Lock and Unlock with <u>different keys</u>!!
- <u>No Secret</u> Key Agreement required

Asymmetric Cryptography



→ **Key Exchange 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 $\mathbf{K_{pub}}$ and $\mathbf{K_{pr}}$ is computed

Public Key Cryptography Applications

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

- **Key Exchange** 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!)

Hybrid Crypto System

In practice, **hybrid system** is used incorporating both asymmetric and symmetric algorithms:

1. **Key exchange** 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**

Hybrid Crypto System – Example

Alice

Bob

$$(K_{pubB}, K_{prB}) = K$$

$$\xleftarrow{\quad K_{pubB} \quad}$$

Choose a random symmetric key K

$$c_1 = e_{K_{pubB}}(K) \xrightarrow{\quad c_1 \quad}$$

$$K = d_{K_{prB}}(c_1)$$

Key Exchange (asymmetric)

message m

$$c_2 = AES_K(m) \xrightarrow{\quad c_2 \quad} m = AES^{-1}_K(c_2)$$

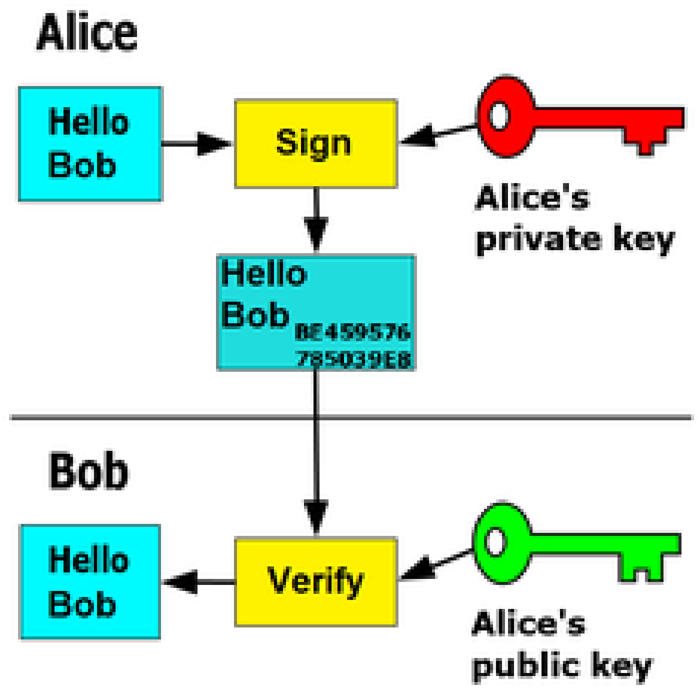
Data Encryption (symmetric)

AES used as the symmetric cipher

Key Exchange with Public Key Crypto

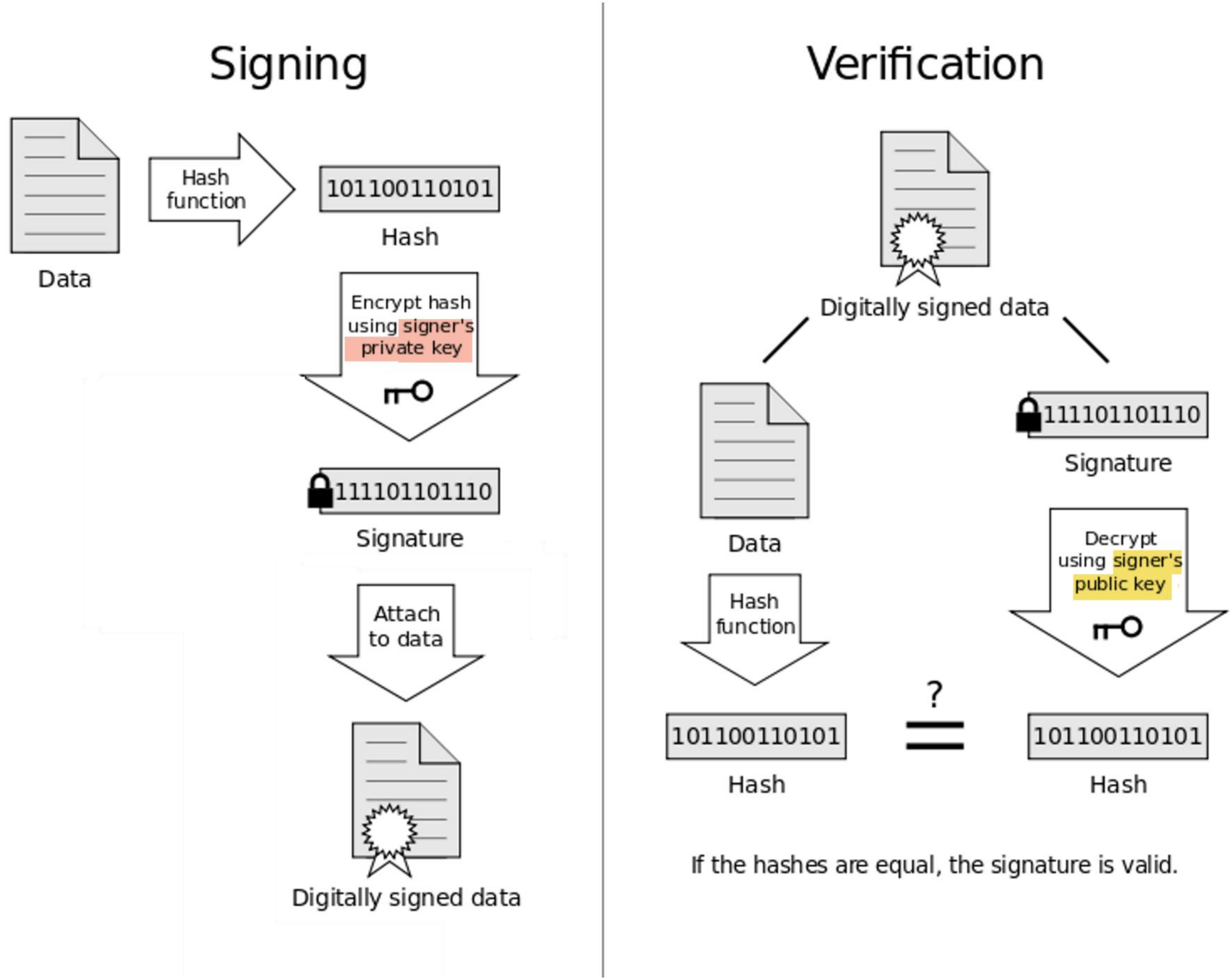
- Alice creates a secret key, encrypts it with Bob's public key and sends it off
- Bob decrypts the message with his private key
- Use shared key for further communication
- This is how many applications work
- Could encrypt/decrypt using public key cryptography but it is slow

Digital Signatures

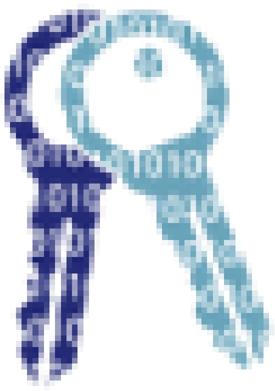


1. Alice signs a message with her private key
2. Bob can verify that Alice sent the message (i.e., non-repudiation) and that the message has not been modified (i.e., integrity)

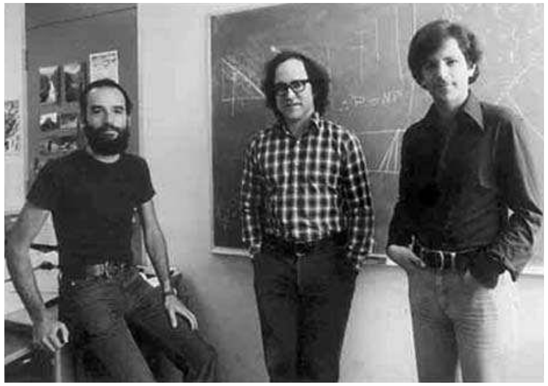
Public-Key Signature Process

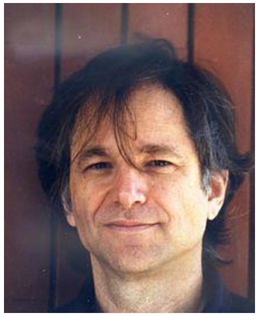


RSA



RSA Cryptosystem





Rivest **Shamir** **Adleman**

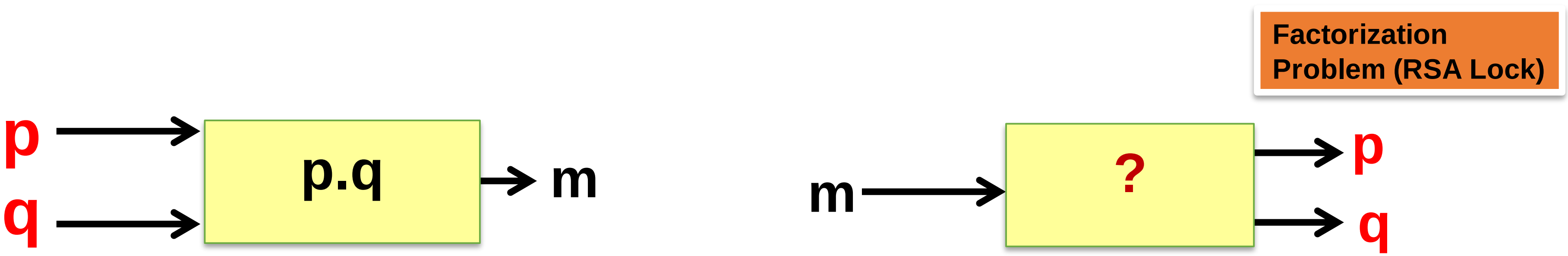
- Developed by Rivest, Shamir, and Adleman in 1977
- Until now, RSA is the most widely use asymmetric cryptosystem although Elliptic Curve cryptography (ECC) becomes increasingly popular
- RSA is mainly used for two applications: Key exchange & Digital signatures

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 Prime_1 and 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

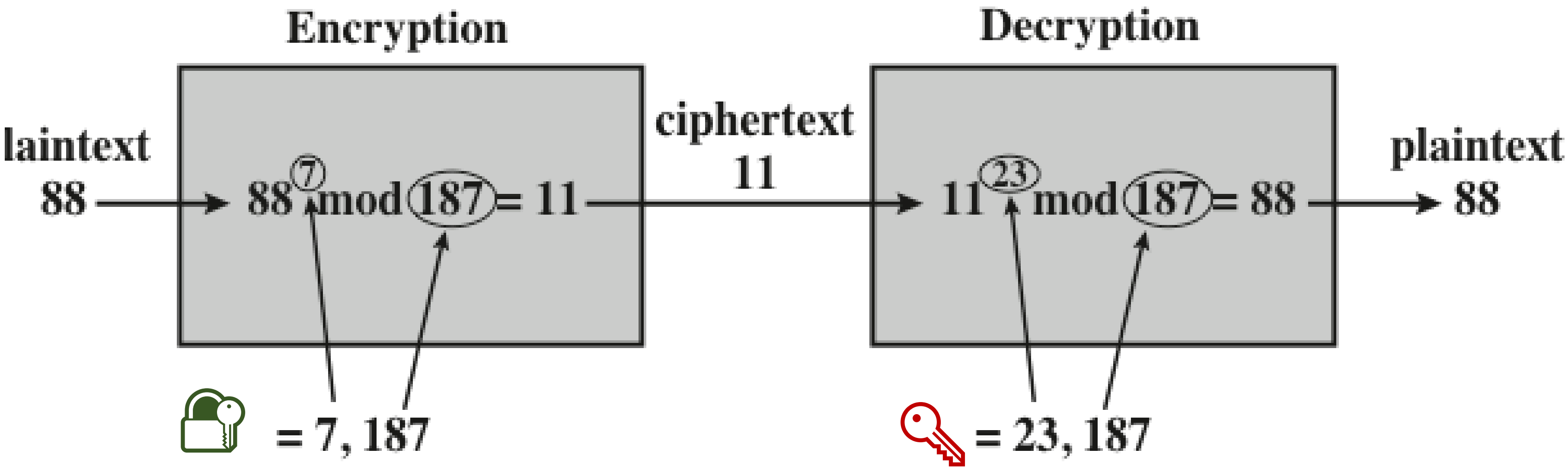
$$c = e_{k_{pub}}(m) = m^{e} \bmod n$$

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

$e_{k_{pub}}()$ 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
= 7, 187
= 23, 187

RSA Key Generation Algorithm

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 *odd* public exponent ***e*** such that $\mathbf{1 < e < \Phi(n)}$ *and* $\mathbf{gcd(e, \Phi(n)) = 1}$ *(i.e. e does not share any factor with Φ(n))*
5. Compute the private key *d* such that $(d * e) \bmod \Phi(n) = 1$

=> Compute *d* by solving $\mathbf{ex + \varphi(n)y = 1}$ using **Extended Euclidean Algorithm**

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

- d is the **inverse** of e mod Φ(n) => $\mathbf{d = e^{-1} \bmod \Phi(n)}$
- gcd$(e, \Phi(n)) = 1$ ensures that ***e*** has an inverse (i.e., a private key *d*)
- **Φ** is called Phi **gcd** is the greatest common divisor

Example: RSA with small numbers

ALICE

Message ***m = 4***

BOB

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. Compute the private key d such that $(d * e) \bmod \Phi(n) = 1$

=> $d=7$

$K_{pub} = (33,3)$

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

c = 31

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

Compute d using Extended Euclidean Algorithm

- EED calculates **x** and **y** such that `ax + by = gcd(a,b)`
  Now let a=e, b=ϕ(n), and thus `gcd(e, ϕ(n))=1`

>> We have to solve: **ex + ϕ(n)y = 1**

We need to solve `3x + 20y = 1`

1. Euclidean algorithm to compute `gcd(3, 20)`	**2. Back substitution** (write 1 as a linear combination of 3 and 20)
20 = 6(3) + 2 3 = 1(2) + 1	1 = 3 – 1(2) 1 = 3 – 1 (20 – 6(3)) 1 = 7(3)-1(20) => **d = 7**

d = x = 7

The value of y does not actually matter, since it will get eliminated by modulo ϕ(n) regardless of its value => can safely discard it

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** (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: $c = m^e \bmod n \rightarrow c = 3^7 \bmod 247 \rightarrow$
  $c = 211$ (ciphertext)
- To decrypt: $m = c^d \bmod n \rightarrow m = 211^{31}$ mode $247 \rightarrow$
  $m = 3$ (plaintext)

Another example

$e*d \bmod \varphi(n) = 1$

$7*d \bmod 40 = 1$

$p = 11$
$q = 5$
$n = 55$
$\varphi(n) = 40$
$e = 7$
$d =$

Step 1: Euclidean algorithm

$40x + 7y = 1$

$40 = 5(7) + 5$

$7 = 1(5) + 2$

$5 = 2(2) + 1$

Step 2: Back substitution

$1 = 5 - 2(2)$

$1 = 5 - 2(7 - 1(5))$

$1 = 3(5) - 2(7)$

$1 = 3(40 - 5(7)) - 2(7)$

$1 = 3(40) - 17(7)$

$d = 40 - 17$
$= 23$

http://igotshittodo.blogspot.com/2016/05/back-substitution-feedback.html

Attacks and Countermeasures

- **Brute force key search**
  - using exhaustive search for factoring of n in order to obtain Φ(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 exchange 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 key should be used for long-term security

References

- Asymmetric cryptography Wikipedia pages

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

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

- Greatest Common Divisor

https://www.youtube.com/watch?v=JUzYl1TYMcU

- Extended Euclidean algorithm

https://www.youtube.com/watch?v=kYasb426Yjk

- Extended Euclidean Algorithm and Inverse Modulo Tutorial

https://www.youtube.com/watch?v=fz1vxq5ts5I