

**EBU6010**

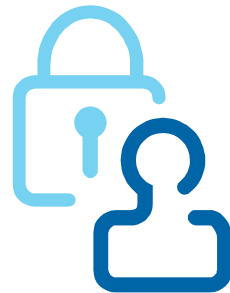
# **Cryptography and Cyber Security**

**Block 2**

**School of Electronic Engineering & Computer Science,**



**Queen Mary**  
University of London



# Recap

- **So far, we looked at ‘Conventional Encryption’**
  - Encryption and decryption use the same key
- Encryption: Historical methods to Standards
- Modes of operations...

# Overview

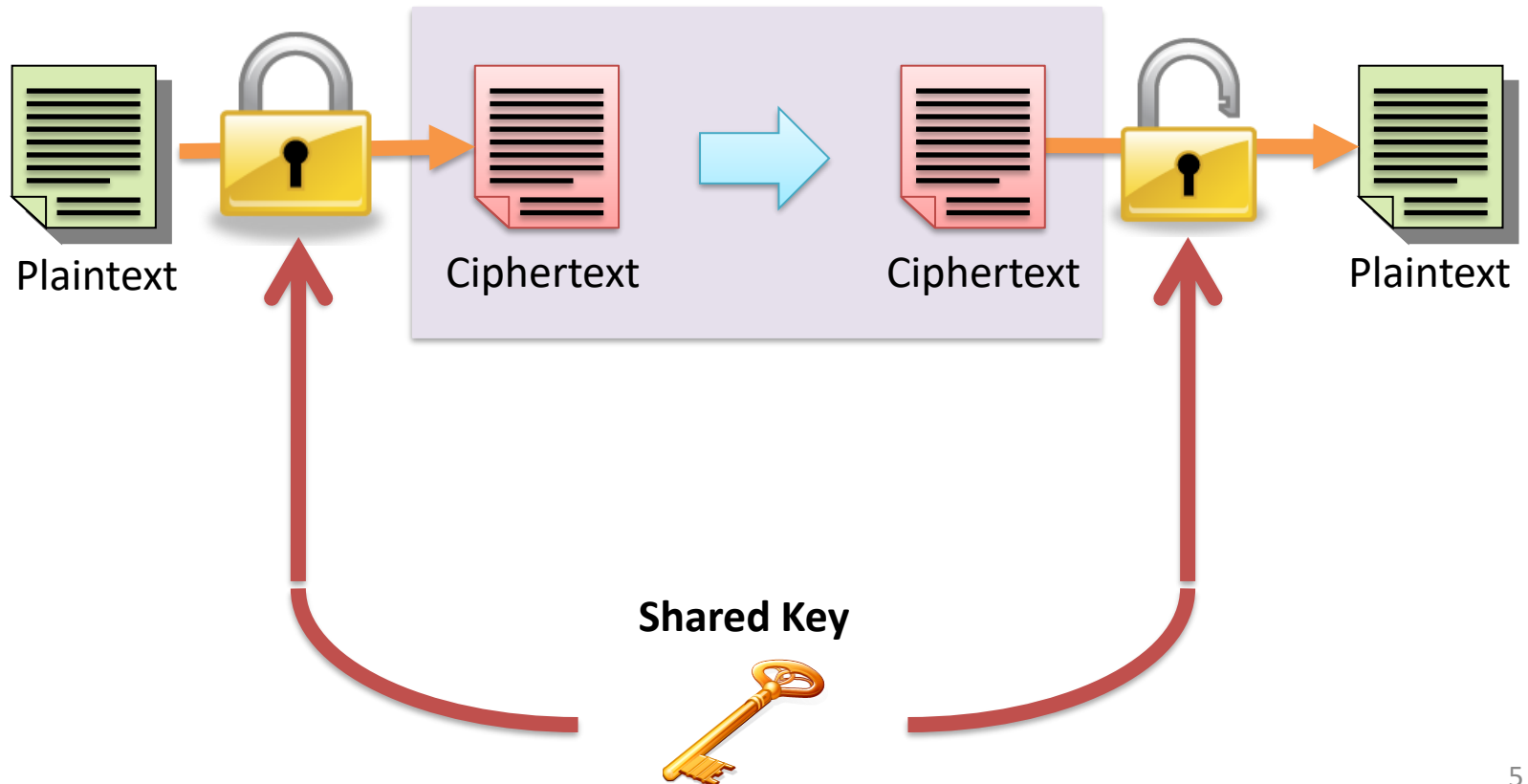
- **Asymmetric encryption**
  - Diffie-Hellman Key Exchange
  - RSA public Key Algorithm
- **Introduction to Certificate**
- **Message Authentication (Data Integrity)**
  - Hash function (SHA1, SHA-512)
  - MAC
  - Birthday Attack
  - HMAC
- **Digital Signature**

# Overview

- **Symmetric (conventional) encryption**
  - Encryption and decryption use the same key
- **Asymmetric encryption**
  - Encryption and decryption use different keys:
    - public key
    - Private key
- **Public-Key is an 'Asymmetric' method**
  - Uses different keys
  - Based on mathematical functions (instead of permutation and substitution methods)
  - It should provide enhanced features:
    - Confidentiality.
    - Key distribution.
    - Authentication,

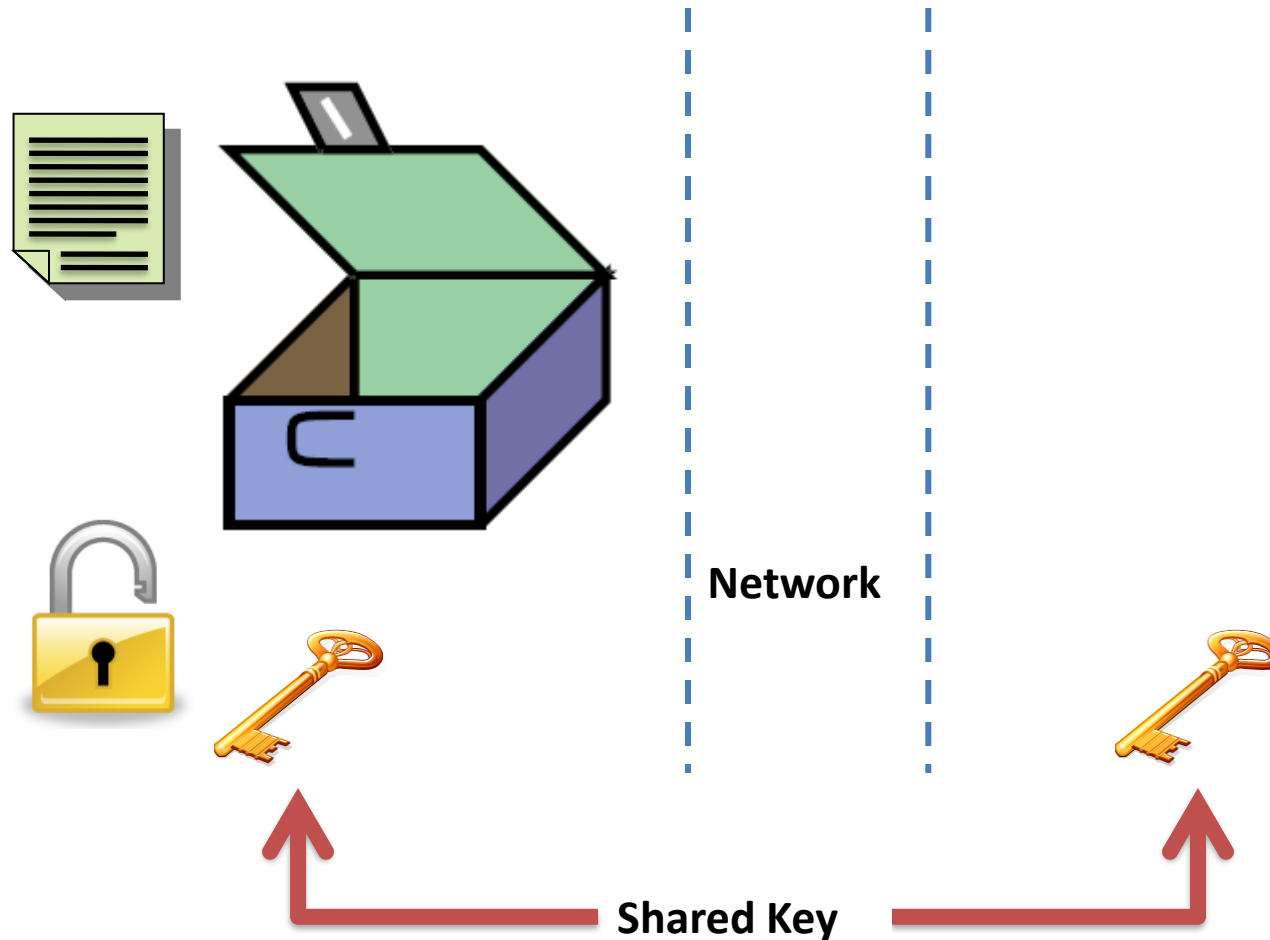
# Conventional Encryption

- Encryption and decryption share the same key



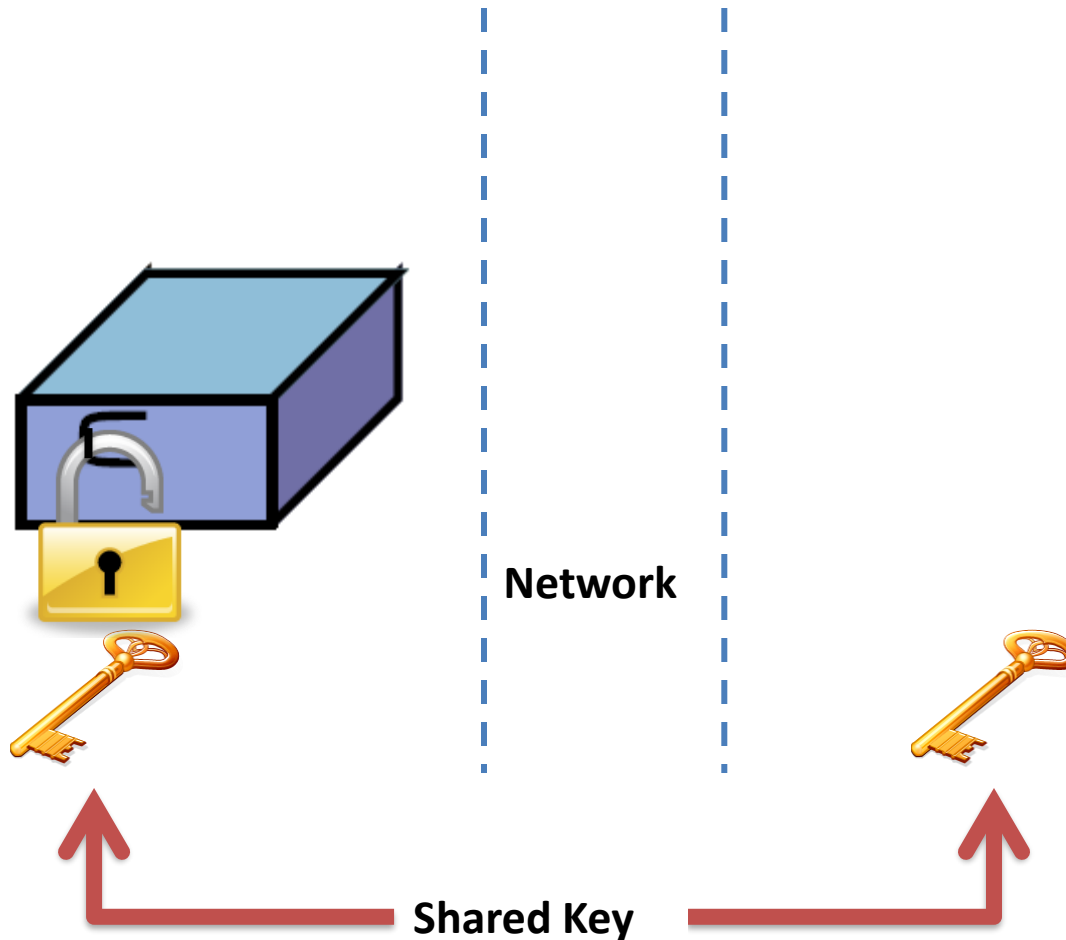
# Conventional Encryption

- Plaintext



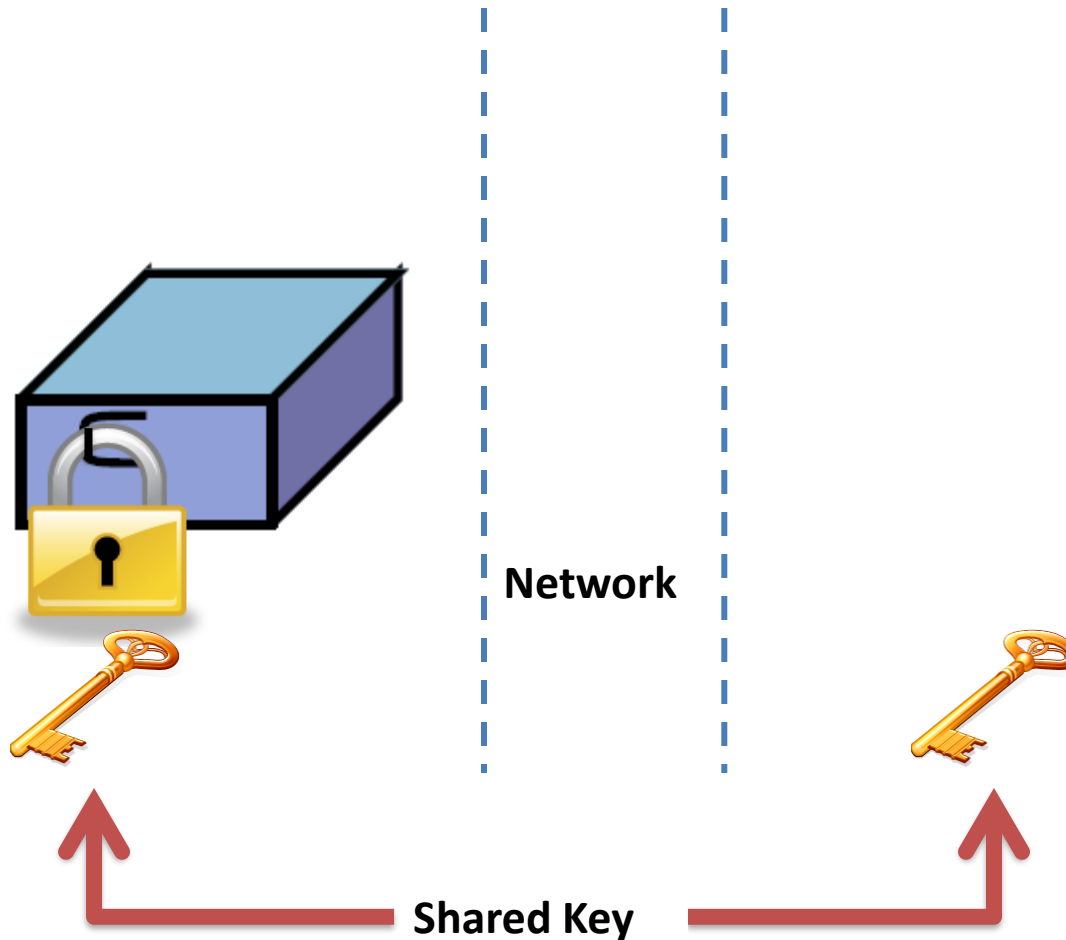
# Conventional Encryption

- Encryption process



# Conventional Encryption

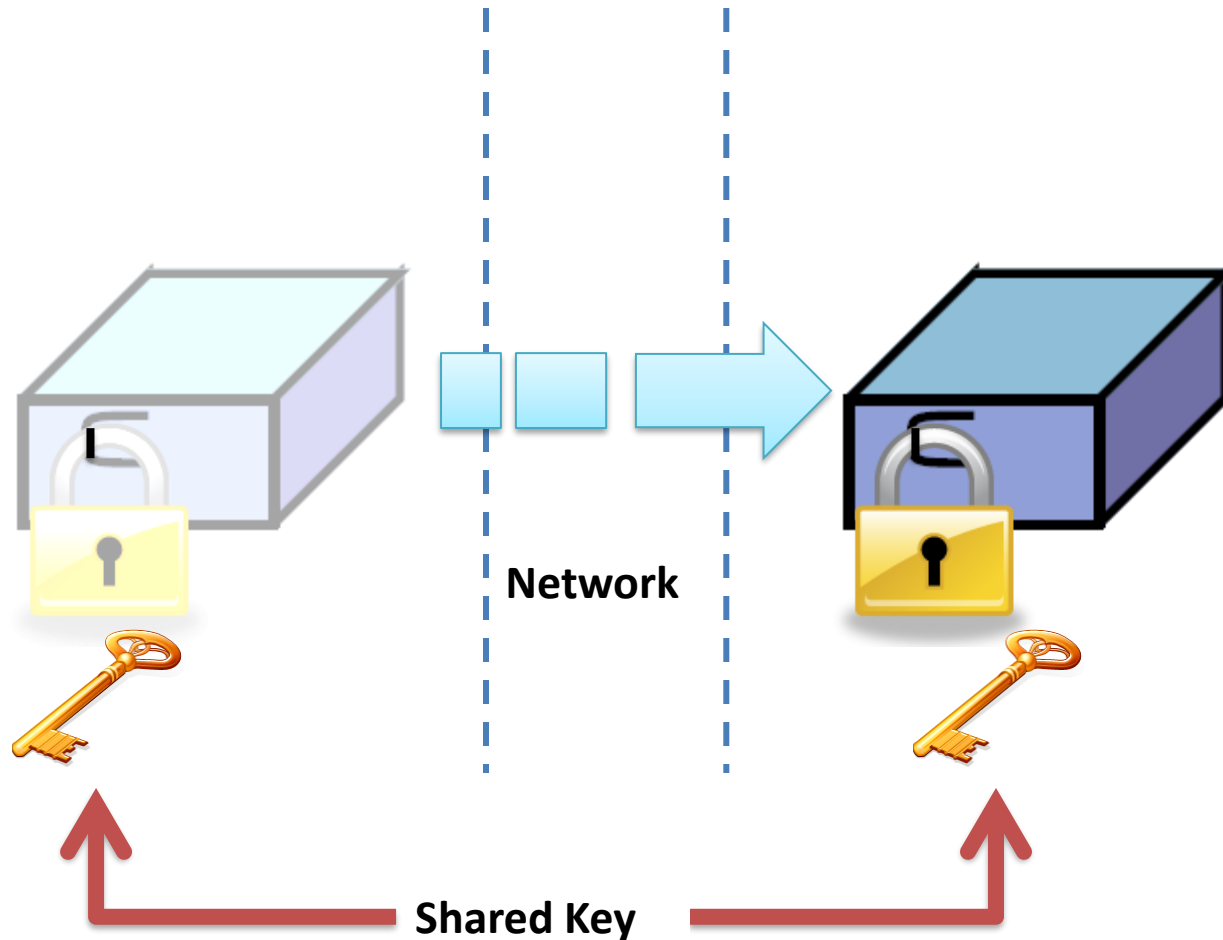
- Ciphertext





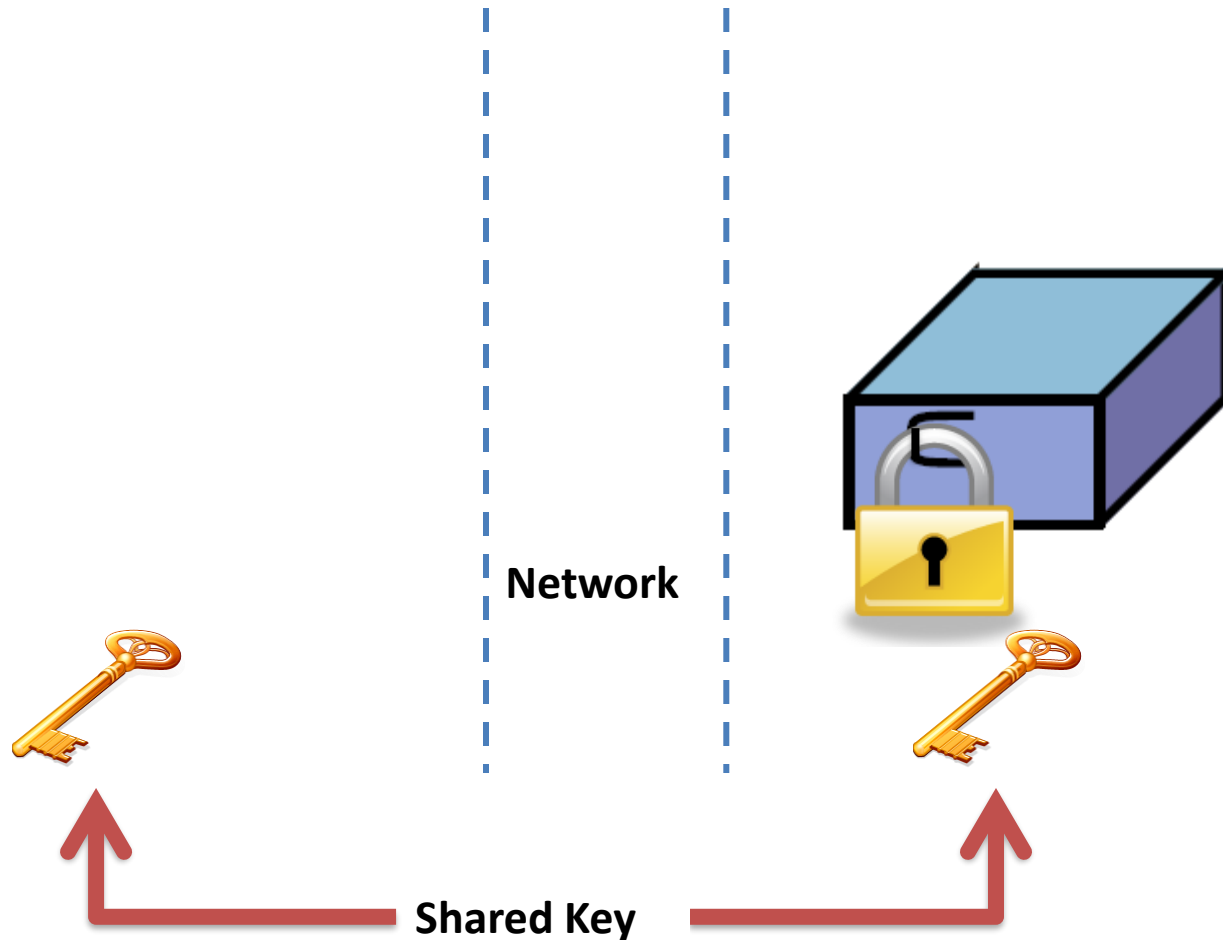
# Conventional Encryption

- Message delivery



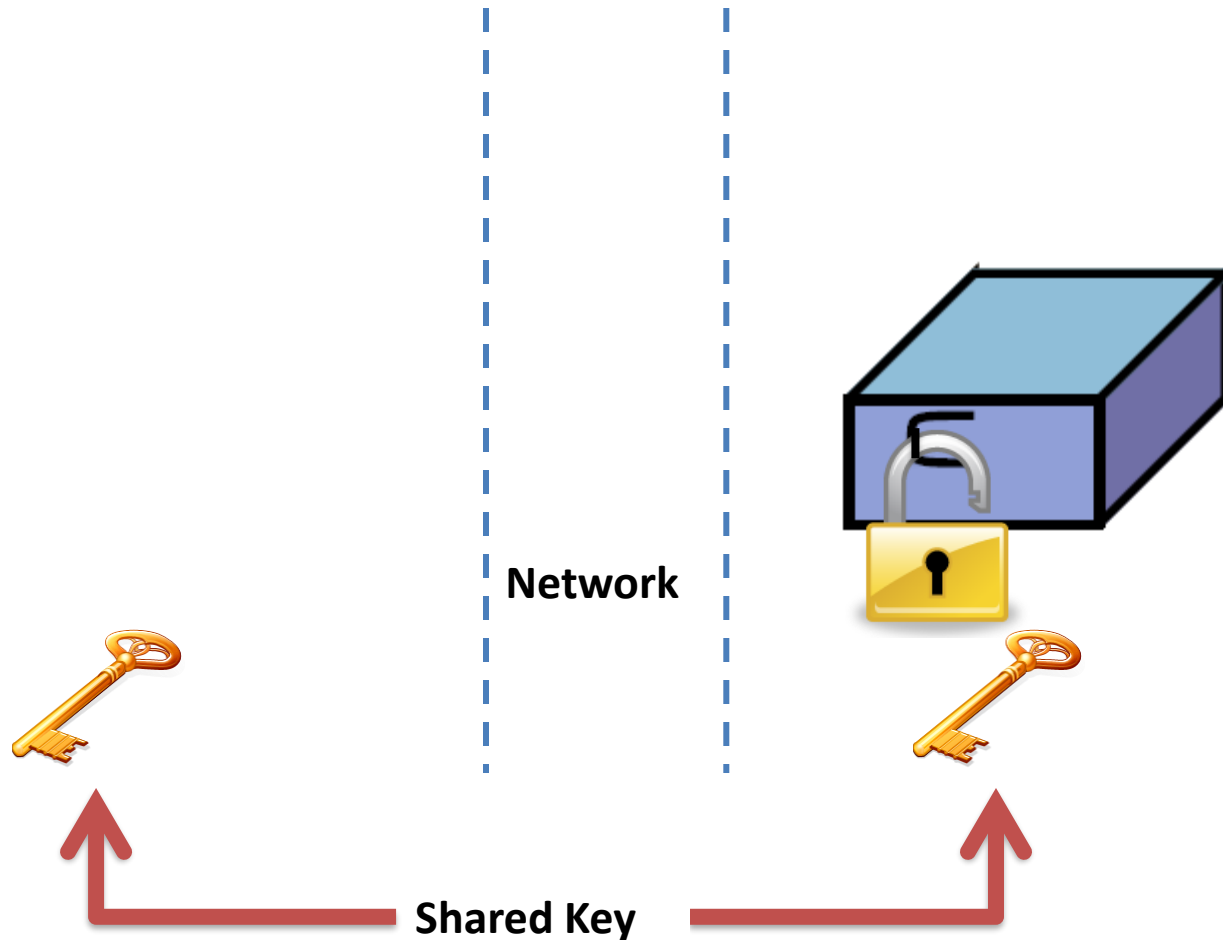
# Conventional Encryption

- Message delivery



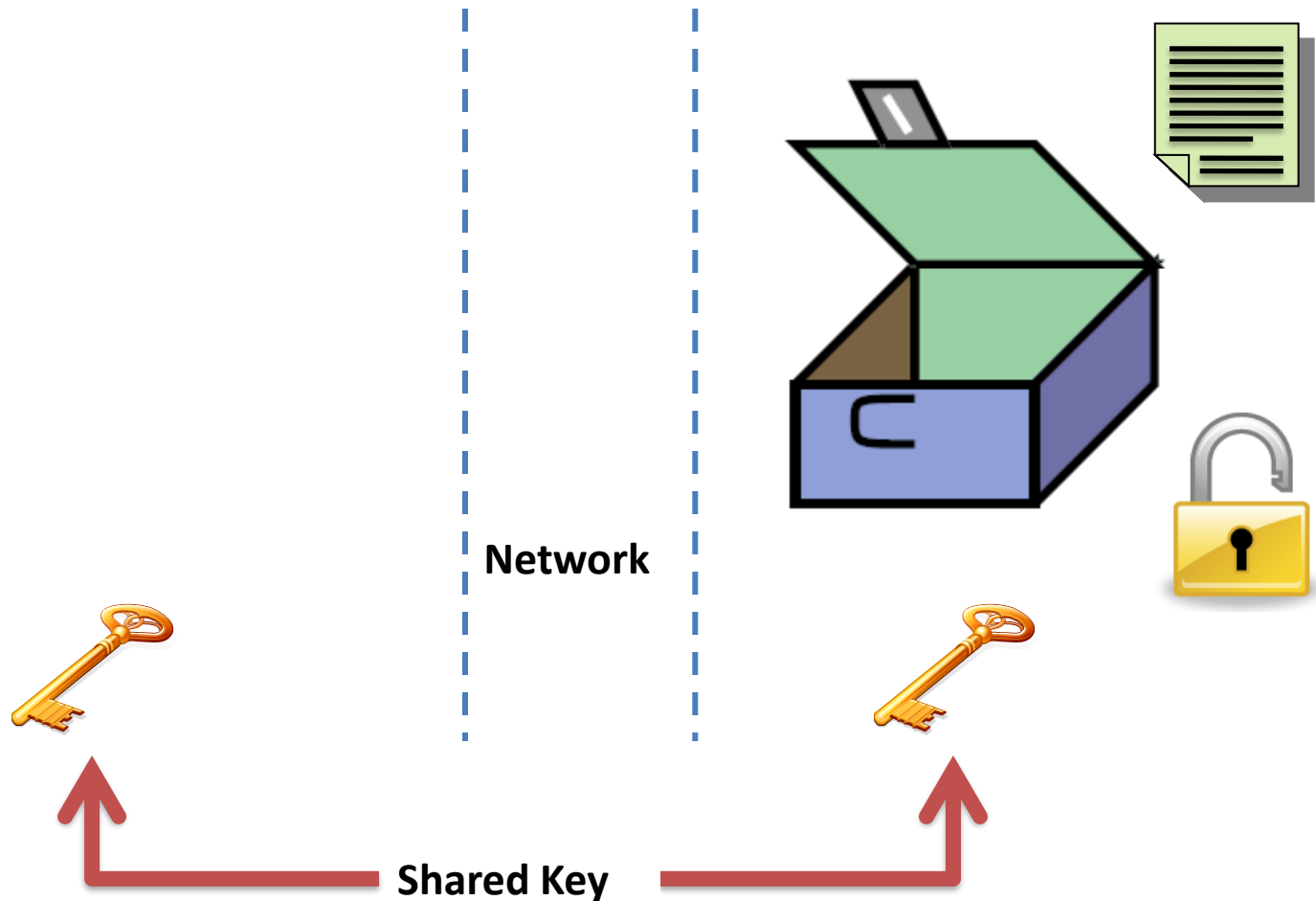
# Conventional Encryption

- Decryption process



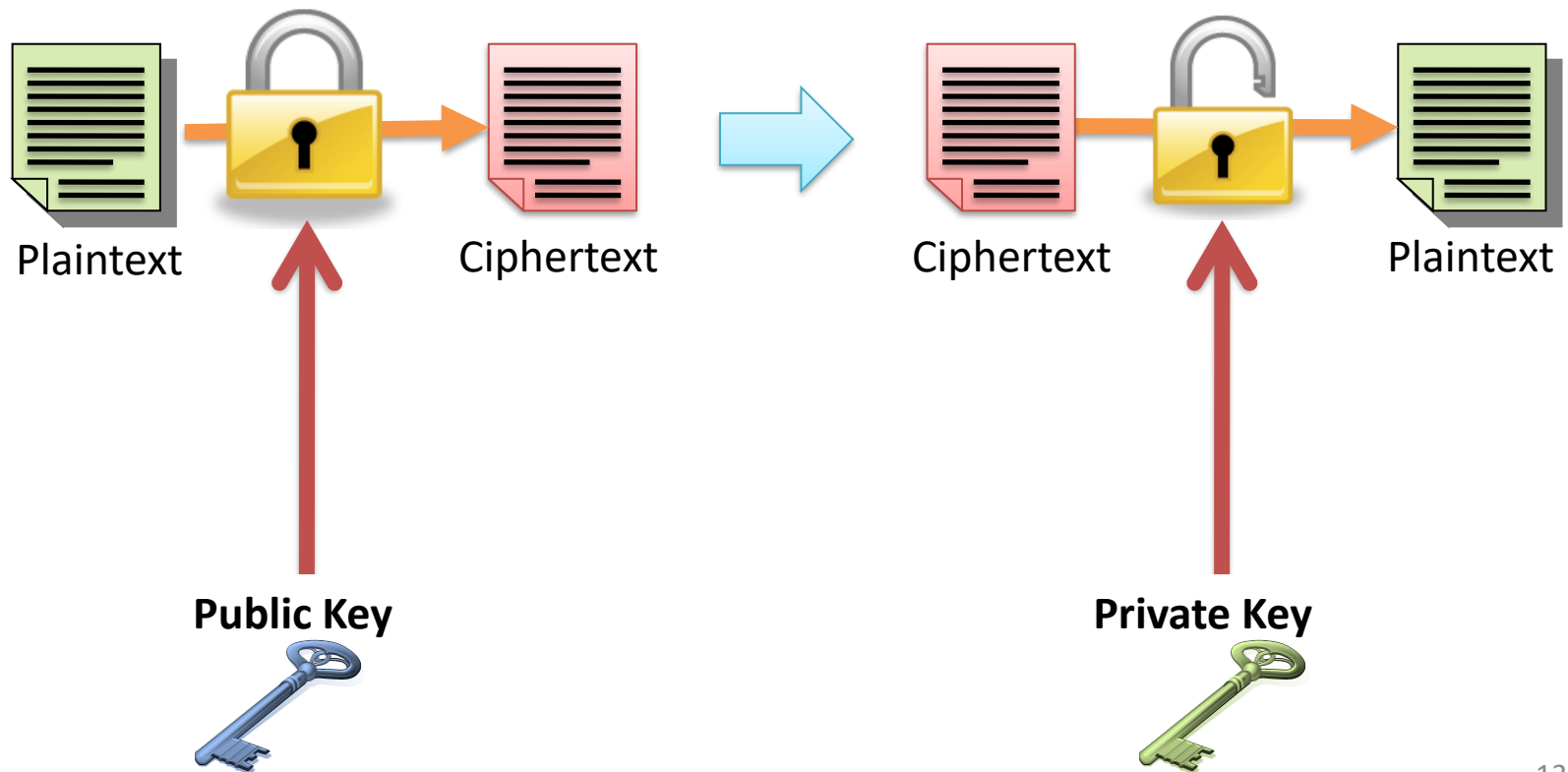
# Conventional Encryption

- Plaintext



# Public-Key Encryption

- Encryption and decryption use different keys



# Public Key Encryption

- 

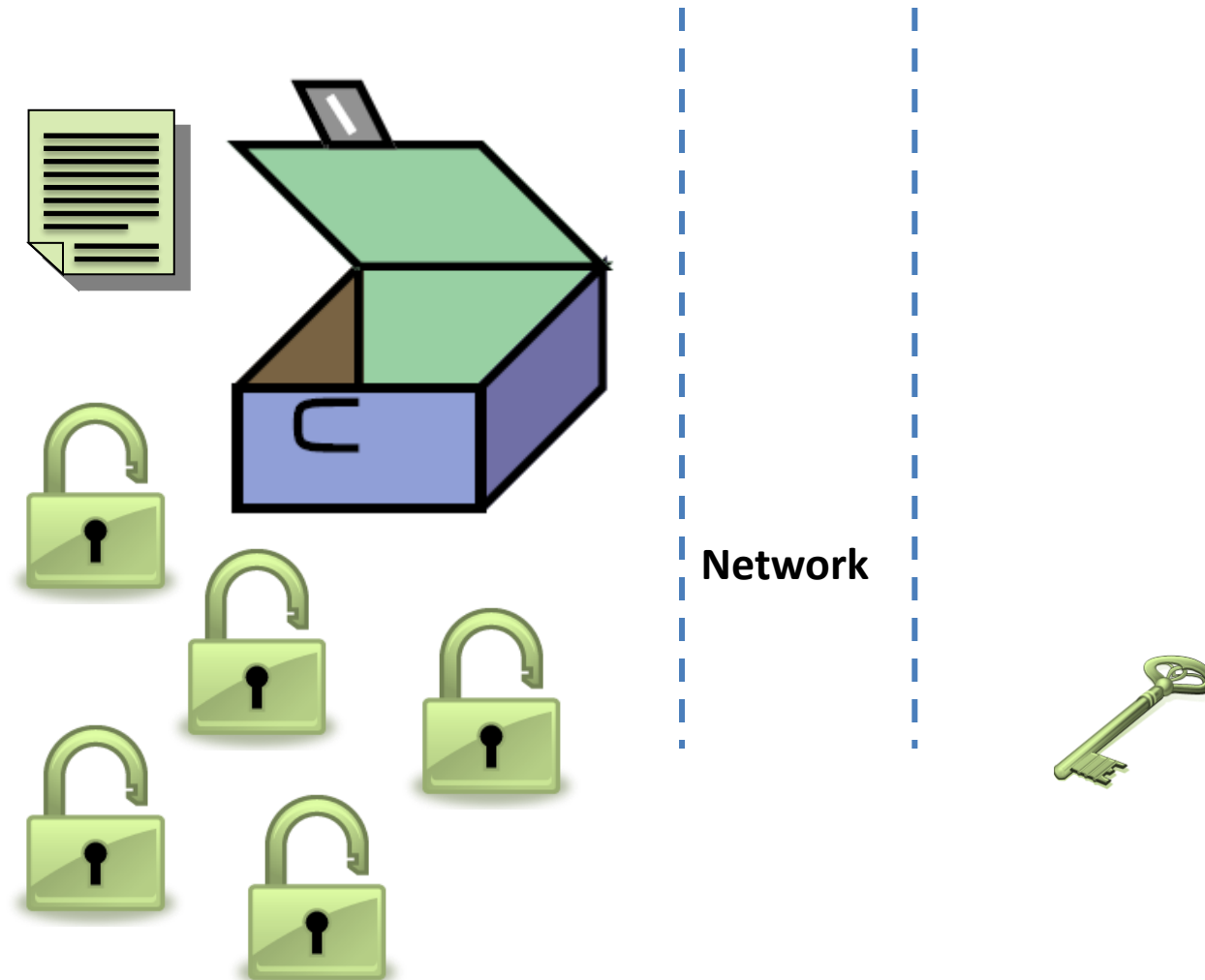


Network



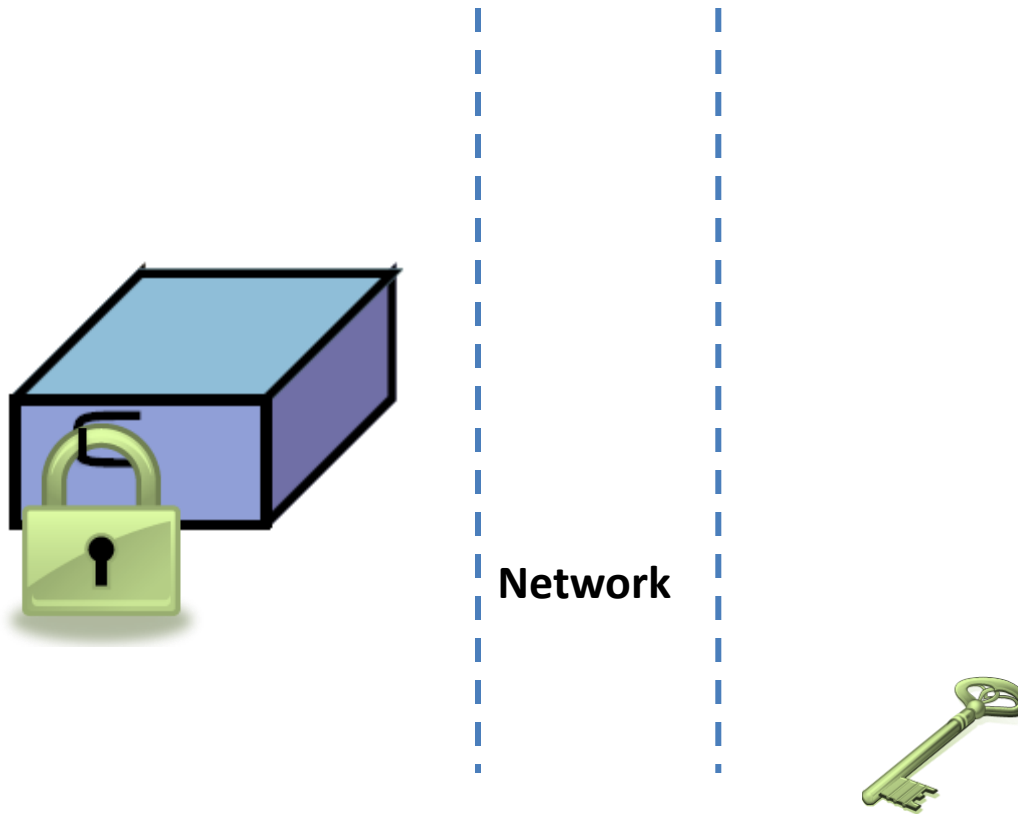
# Public Key Encryption

- 



# Public Key Encryption

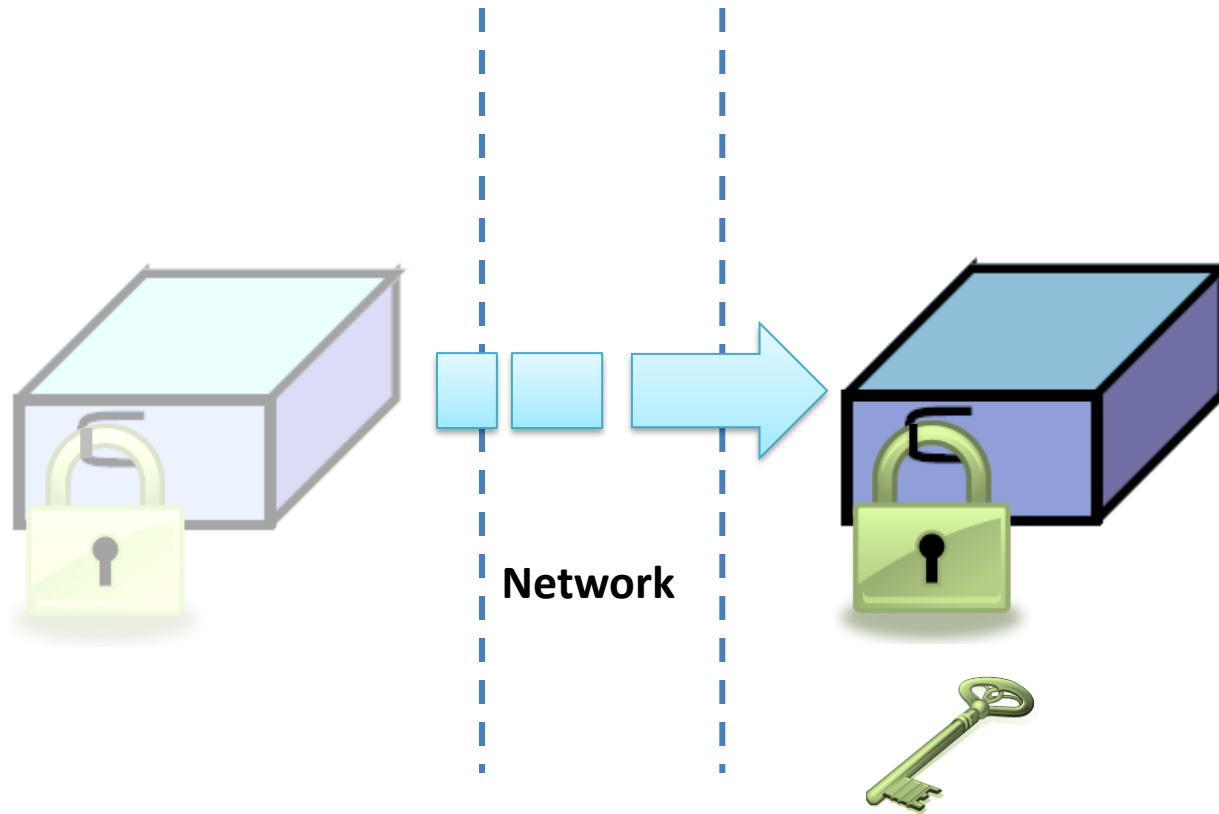
- 





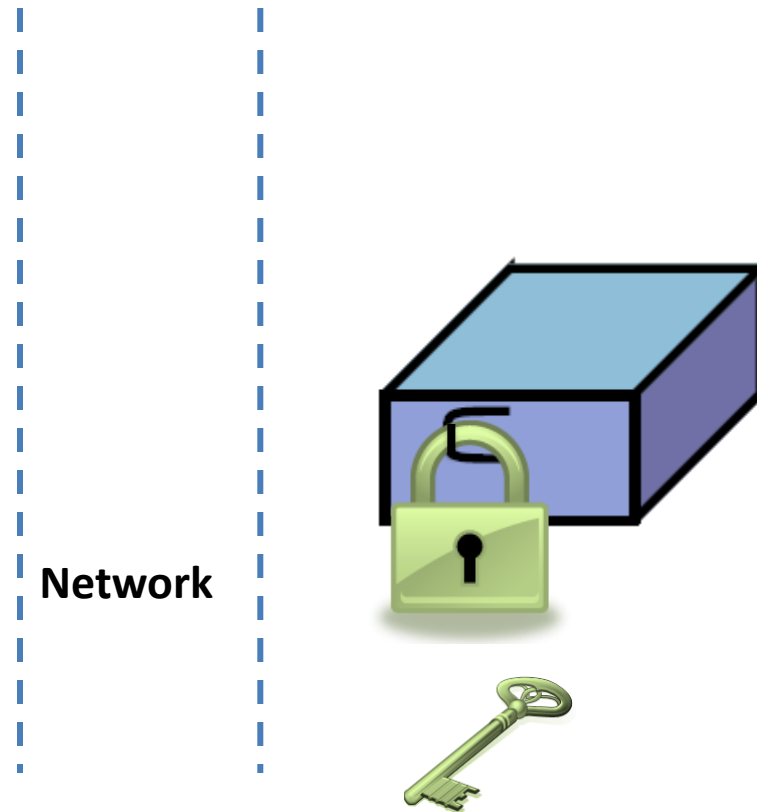
# Public Key Encryption

- 



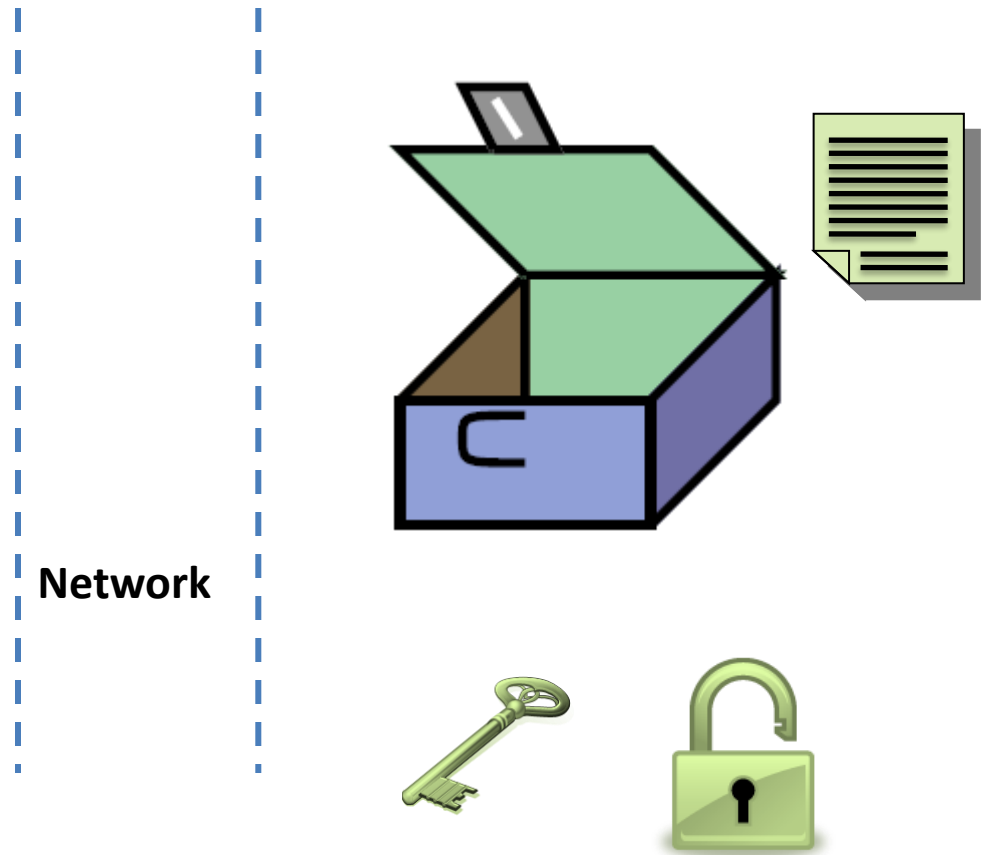
# Public Key Encryption

- 



# Public Key Encryption

- 



# Advantages vs Disadvantages

- **Advantages**

- No need to share the same key

- **Disadvantages**

- How to trust who's the lock's holder?
- Who originated the document?

# Public-Key Encryption

- **Requirements:**
  - Easy to generate pair of key.
  - If **A** knows the public key of **B** it is easy to generate the ciphertext **c** from the original plaintext message **m**.
  - It is easy for **B** to decrypt the message using the private key.
  - It is computationally infeasible for an opponent, knowing the ciphertext and public key to recover the original message **m**.

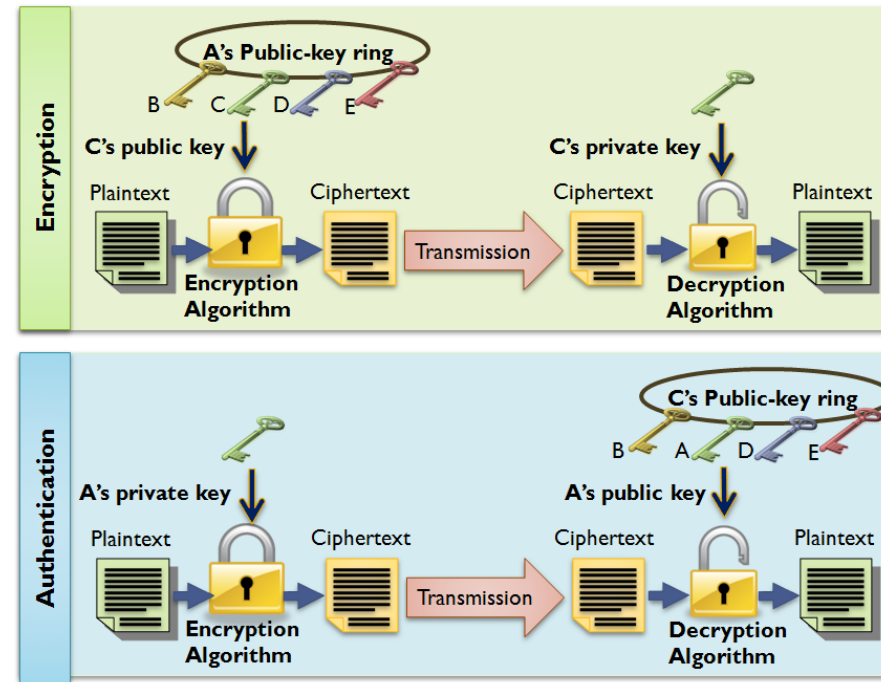
# Public-Key Encryption

- **Basic elements:**

- Plaintext
- Encryption Algorithm
- Public and Private keys
- Ciphertext
- Decryption Algorithm

- **Operation principle**

1. Each user generate two keys, public and private.
2. The user put their public key in public register. These public keys are accessible by anyone.
3. If User1 wishes to send a message to User2, User1 encrypts the message using User2's public key.
4. User2 decrypts it using his/her own private key. No one else could decrypt the message because it's kept privately.





# **Public Key Encryption**

## (Asymmetric Cryptography)

# Public-Key Algorithms

- *Diffie-Hellman* key exchange
- RSA public-key encryption algorithm



Rivest, Shamir and Adleman

- Digital Signature Standard (DSS)
- Elliptic-Curve Cryptography



# One Way Functions

- A one-way mathematical function is very easy to do, but very difficult to reverse.

*Examples*      Easy to do:

$$7919 \times 7927 = 62\,773\,913$$

Difficult:

What are the factors of      1 689 259 081 189

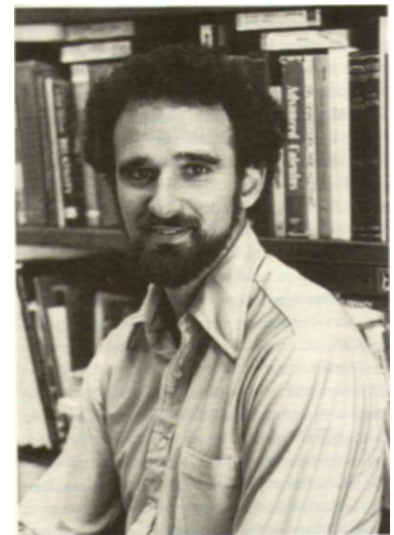
- Public-key encryption can be thought of as a function that anyone should be able to compute, since the encryption key is public, and
- It should be very difficult for an attacker to efficiently determine a plaintext from knowledge of a ciphertext and the public encryption key.

# Examples of the difficulty of factoring the product of two primes

Challenge number	Difficulty of factoring to the two primes
15	Everyone can do this instantly
143	Doable with a little thought
6887	Should not take more than a few minutes
31897	A calculator is now useful
20-digit number	A computer is now required
600-digit number	This is impossible in practice
600-digit even number	One factor immediate, other easily computed
600-digit number with small factor	One factor easily found, other easily computed

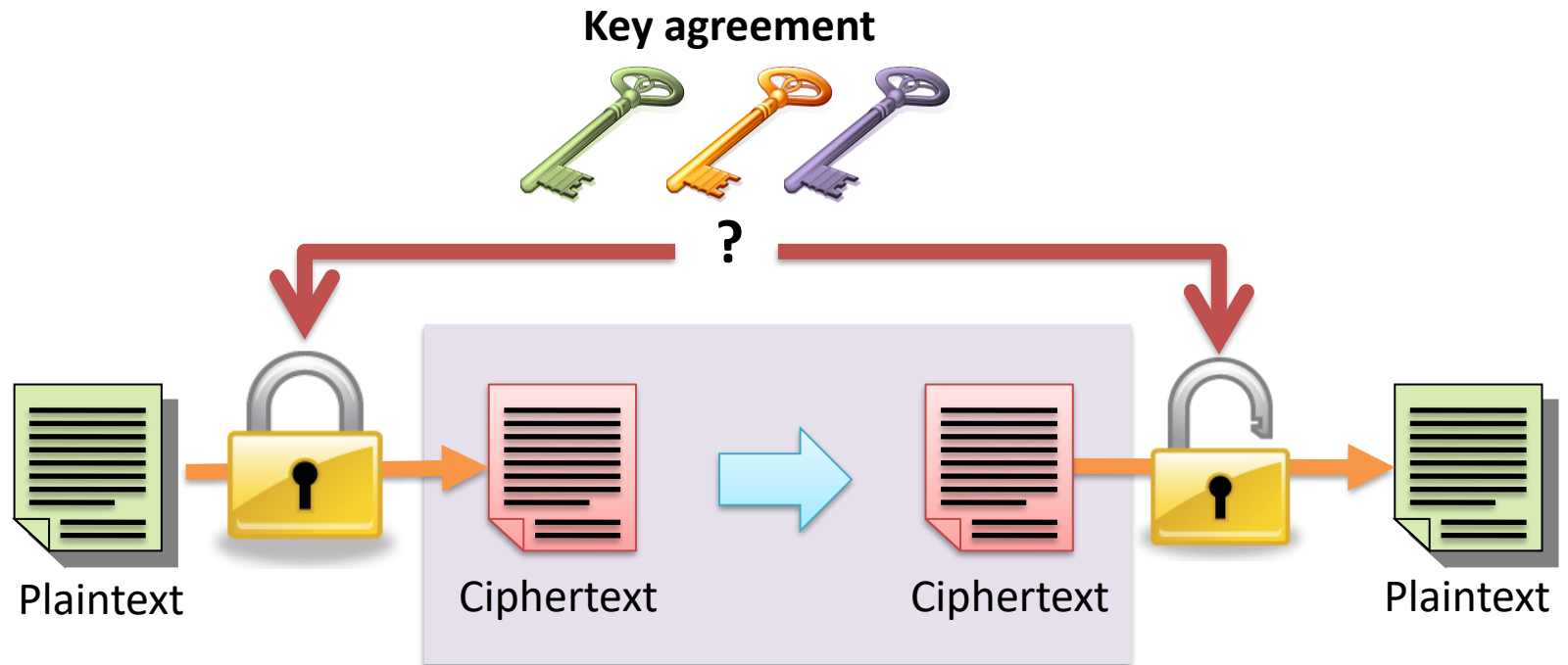
# Public-Key Encryption

- Proposed by *Diffie* and *Hellman* in 1976
- Based on mathematical functions
- Used in:
  - Confidentiality
  - Key distribution
  - Authentication



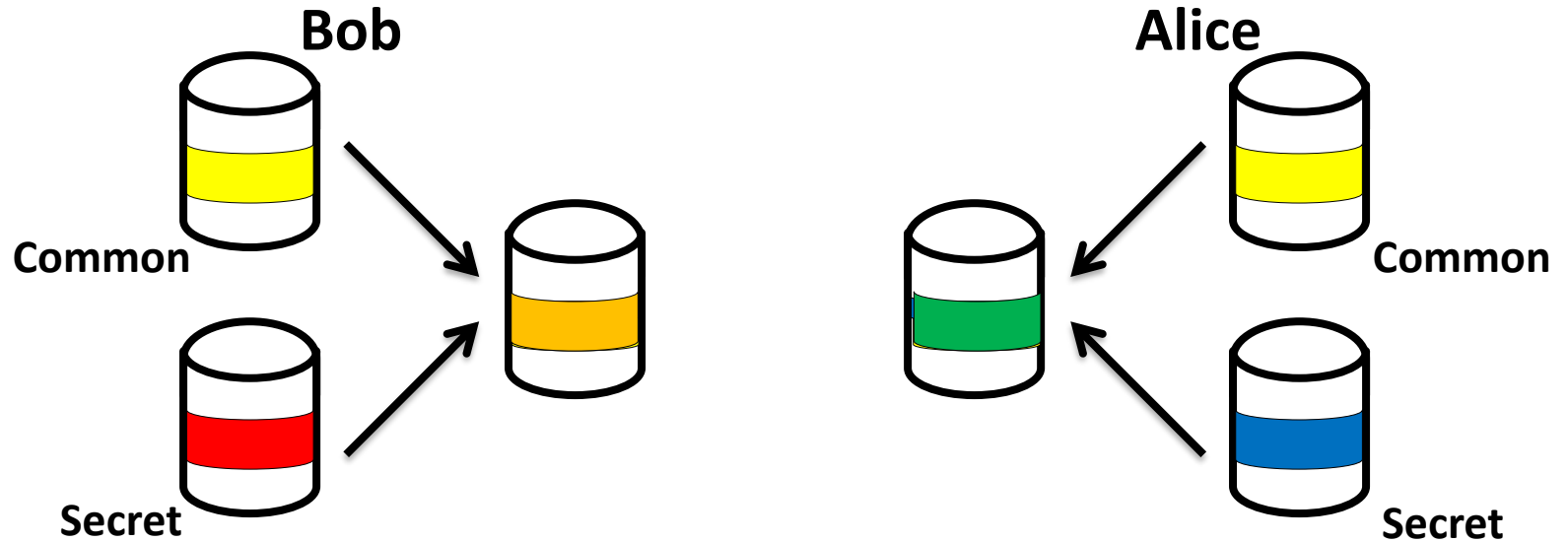
*Diffie* and *Hellman*

# Diffie-Hellman Key Exchange

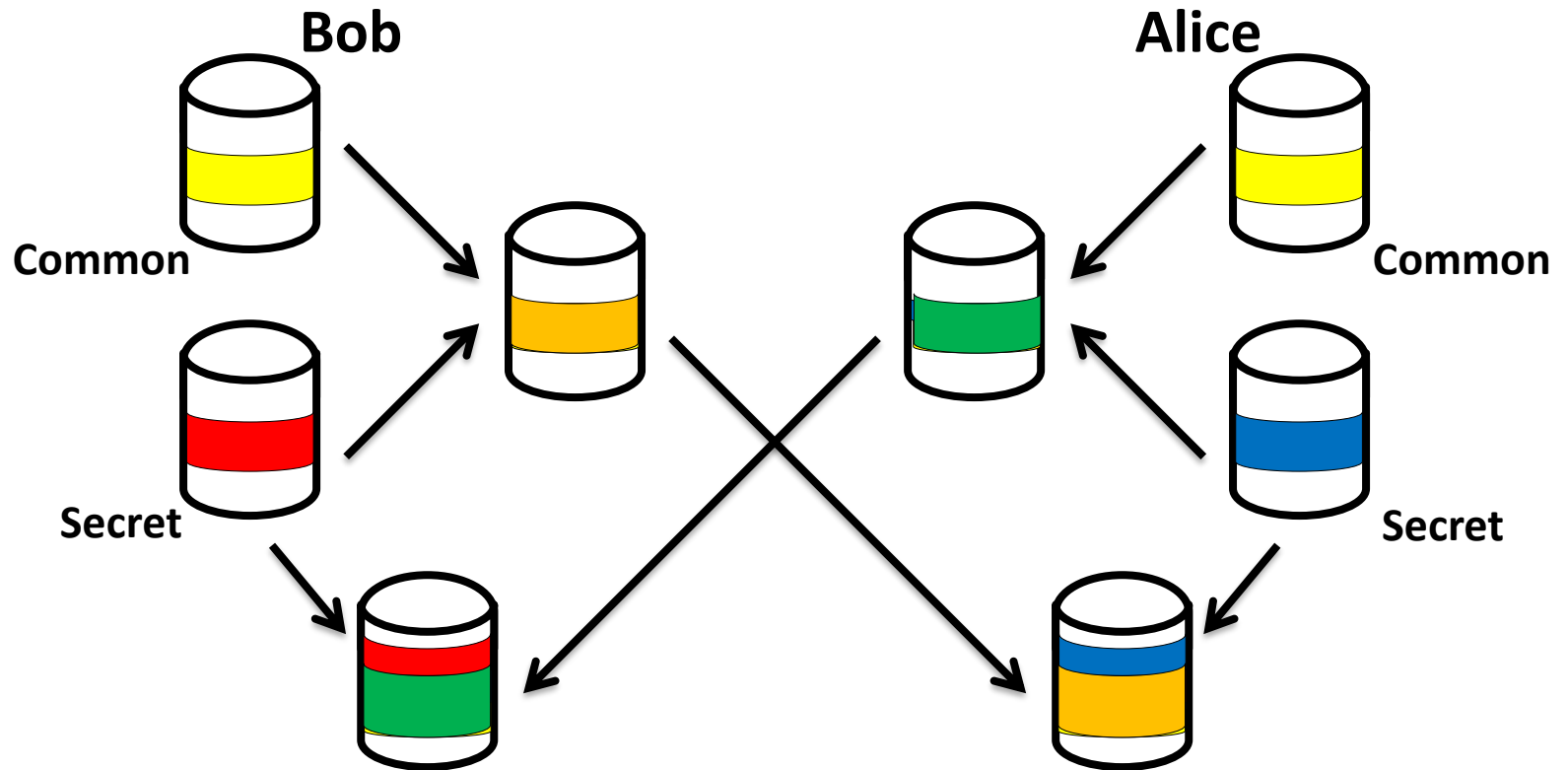


- The algorithm was developed to enable two users to exchange a secret key securely. The algorithm itself is limited to the exchange of keys.
- Based on the difficulty to compute discrete logarithms.

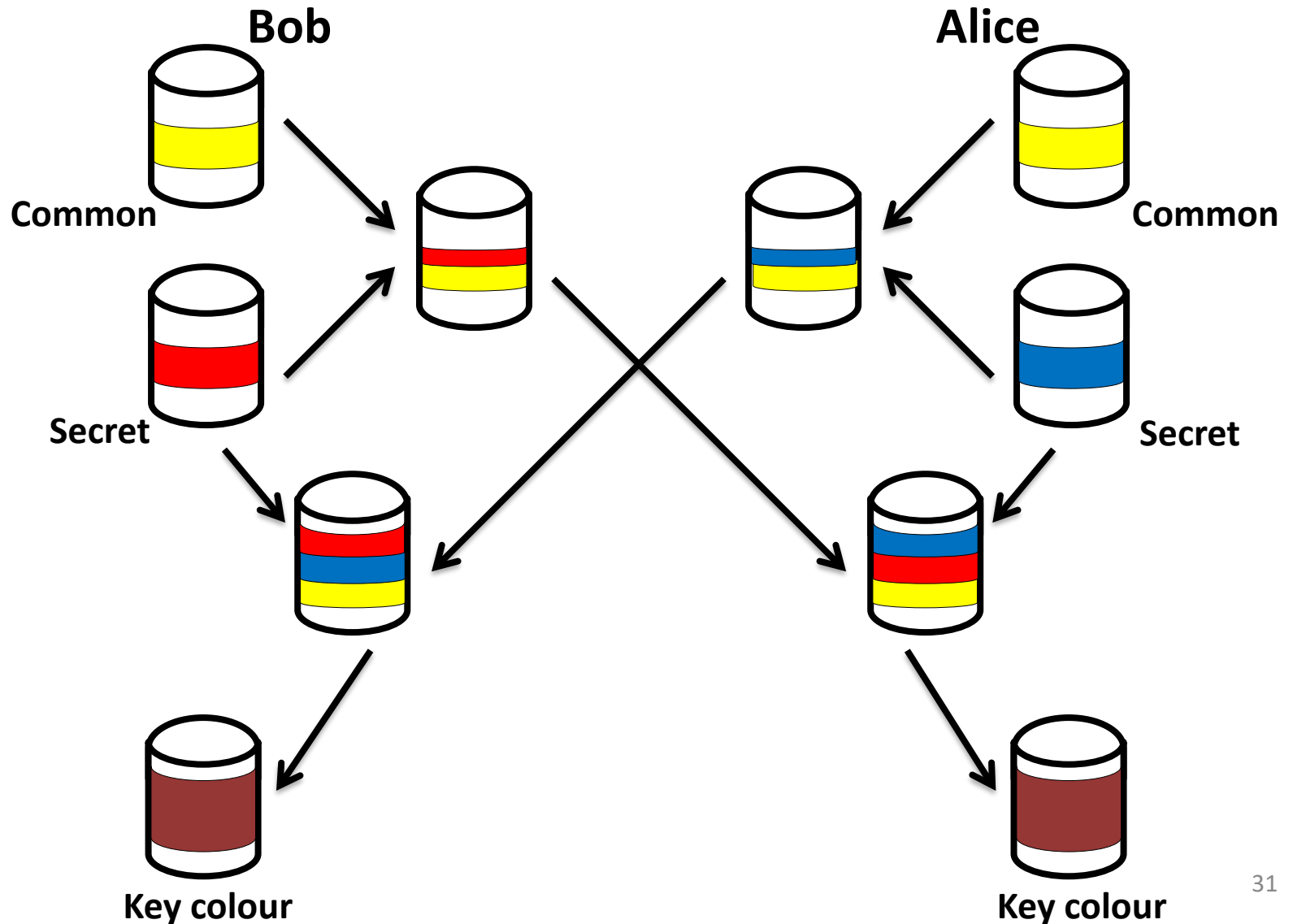
# Analogy: The secret colour



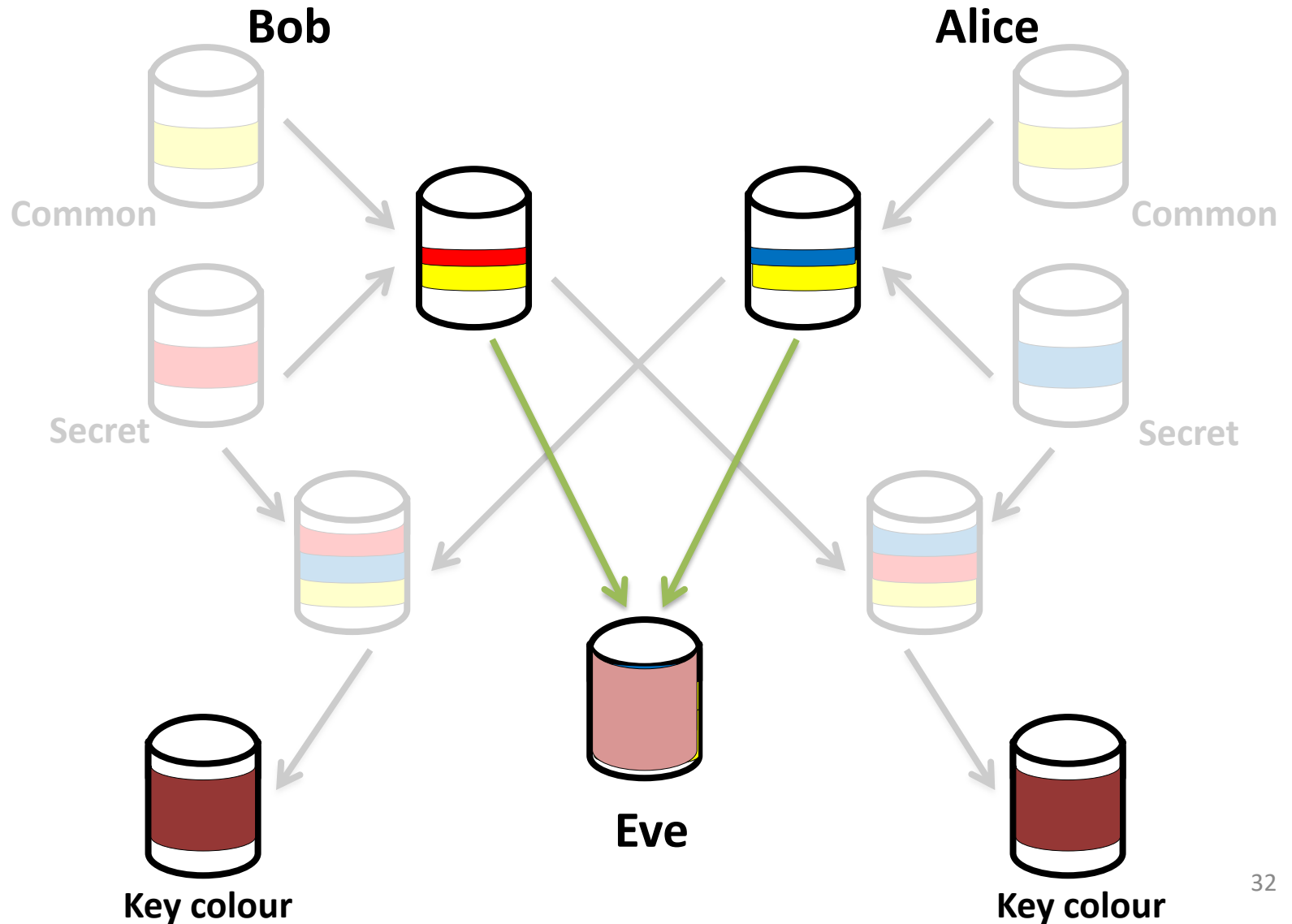
# Analogy: The secret colour



# Analogy: The secret colour



# Analogy: The secret colour





# Modular Arithmetic

Modular arithmetic is a system of arithmetic for integers, where numbers "wrap around" upon reaching a certain value;

$$a = n \times q + r$$

where  $r$  is the remainder or  $r = a \bmod q$  &  $n \in \mathbb{I}$

e.g.  $14 \bmod 12 = 2 \rightarrow 14 = 1 \times 12 + 2$   
 $26 \bmod 12 = 2 \rightarrow 26 = 2 \times 12 + 2$

*A good example is the clock!*



Since the hour number starts over after it reaches 12, this is arithmetic modulo 12.

12 is **congruent** not only to 12 itself, but also to 0, so the time called "12:00" could also be called "0:00", since  $0 \equiv 12 \bmod 12$

So, what is the Congruent? →

# Discrete Logarithm

- **Definition**

- If  $a$  is a primitive root of the prime number  $p$  then

$$a \bmod p, a^2 \bmod p, \dots, a^{p-1} \bmod p,$$

are distinct and consist of the integers 1 through  $p-1$  in some permutation.

- **Example**

- If  $p = 71$  and  $a = 7$  The discrete logarithm, is the exponent  $i$  such that  $b = a^i \bmod p$  where  $0 < i \leq p-1$  and  $a$  is a primitive root of  $p$ .

# Discrete Logarithm example

– Evaluate  $7^i \bmod 71$

7	49	59	58	51	2	14	27	47	45	31	4	28	54	23	19
62	8	56	37	46	38	53	16	41	3	21	5	35	32	11	6
42	10	70	64	22	12	13	20	69	57	44	24	26	40	67	43
17	48	52	9	63	15	34	25	33	18	55	30	68	50	66	36
39	60	65	29	61	1										

– All different because 7 is a primitive root of 71. Try  $8^i \bmod 71$

8	64	15	49	37	12	25	58	38	20	18	2	16	57	30	27
3	24	50	45	5	40	36	4	32	43	60	54	6	48	29	19
10	9	1	8	64	15	49	37	12	25	58	38	20	18	2	16
57	30	27	3	24	50	45	5	40	36	4	32	43	60	54	6
48	29	19	10	9	1										

# Diffie-Hellman algorithm

## Global Public Elements

$q$	Prime number
$\alpha$	$\alpha < q$ , $\alpha$ a primitive root of $q$

## User Key generation

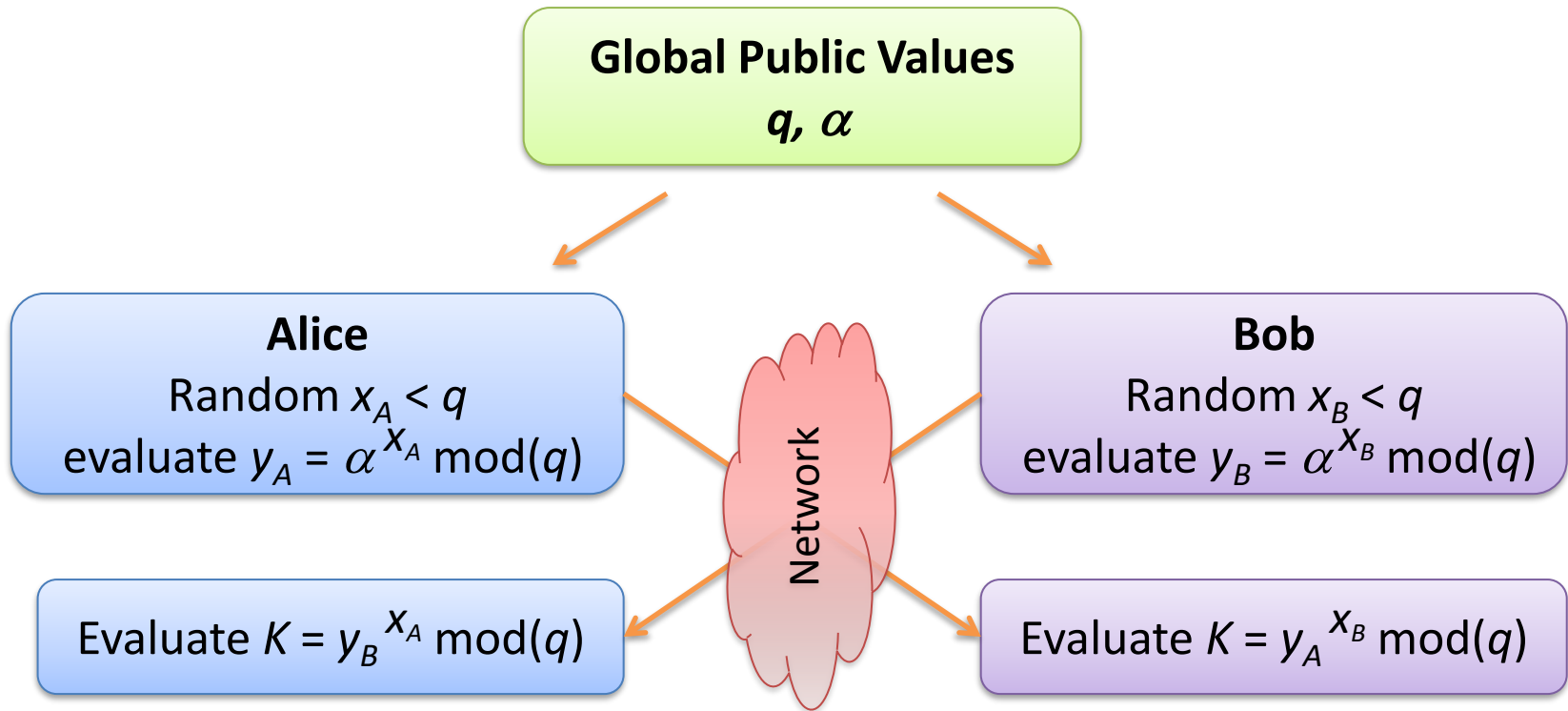
### Alice

Select a private $x_A$	$x_A < q$
Calculate public $y_A$	$y_A = \alpha^{x_A} \bmod q$

### Bob

Select a private $x_B$	$x_B < q$
Calculate public $y_B$	$y_B = \alpha^{x_B} \bmod q$

# Diffie-Hellman algorithm



# ***Diffie-Hellman*** algorithm

- Alice generates a private key  $x_A$ , evaluates  $y_A$  and sends that to Bob.
- Bob responds by generating a private key  $x_B$ , evaluates  $y_B$  and sends that to Alice.
- Both users generate the key  $k$ .

# *Diffie-Hellman* algorithm

- **Generation of Secret Key**

---

<b>Alice</b>	$y_B^{x_A} \bmod q$
--------------	---------------------

<b>Bob</b>	$y_A^{x_B} \bmod q$
------------	---------------------

---

- It works because  $k = y_A^{x_B}$  and  $k = y_B^{x_A}$  produce identical results.

# *Diffie-Hellman* algorithm

- **Example**

- $q = 353$  and  $\alpha = 3$ .
- Alice's secret key  $X_A = 97$ ,
- Bob's secret key  $X_B = 233$
- Alice:  $Y_A = 3^{97} \bmod 353 = 40$
- Bob:  $Y_B = 3^{233} \bmod 353 = 248$

- **Key:**

- Alice:  $248^{97} \bmod 353 = 160$
- Bob:  $40^{233} \bmod 353 = 160$





**... there is an attack on key exchange to consider ..**

# Man-in-the-middle attack

1. Eve prepares for the attack by generating two random private keys  $X_{E1}$  and  $X_{E2}$  and then computing the corresponding public keys  $Y_{E1}$  and  $Y_{E2}$ .
2. Alice transmits  $Y_A$  to Bob.
3. Eve intercepts  $Y_A$  and transmits  $Y_{E1}$  to Bob. Eve also calculates  $K_2 = y_A^{X_{E2}} \bmod(q)$ .
4. Bob receives  $Y_{E1}$  and calculates  $K_1 = y_{E1}^{X_B} \bmod(q)$ .
5. Bob transmits  $Y_B$  to Alice.
6. Eve intercepts  $Y_B$  and transmits  $Y_{E2}$  to Alice. Eve calculates  $K_1 = y_B^{X_{E1}} \bmod(q)$ .
7. Alice receives  $Y_{E2}$  and calculates  $K_2 = y_{E2}^{X_A} \bmod(q)$ .

At this point, Bob and Alice think that they share a secret key, but instead Bob and Eve share secret key and Alice and Eve share secret key.

# Man-in-the-middle attack

Global Public Values  
 $q, \alpha$

**Alice**

Random  $x_A < q$   
Evaluate  $y_A = \alpha^{x_A} \bmod(q)$

**Bob**

Random  $x_B < q$   
Evaluate  $y_B = \alpha^{x_B} \bmod(q)$

**Eve**

Random  $x_{E1} < q$  &  $x_{E2} < q$

**Evaluate**

$y_{E1} = \alpha^{x_{E1}} \bmod(q)$

&

$y_{E2} = \alpha^{x_{E2}} \bmod(q)$

Evaluate  $K = y_B^{x_A} \bmod(q)$   
Evaluate  $K_2 = y_{E2}^{x_A} \bmod(q)$

Evaluate  $K = y_A^{x_B} \bmod(q)$   
Evaluate  $K_1 = y_{E1}^{x_B} \bmod(q)$

**Eve also has**

$K_2 = y_A^{x_{E2}} \bmod(q)$  &  $K_1 = y_B^{x_{E1}} \bmod(q)$



# **RSA** public key algorithm

# RSA public Key Algorithm



- The RSA algorithm was published by Rivest, Shamir and Adleman in 1977
- Invented independently at the u.k. Government Communications HeadQuarters (GCHQ) earlier in 1973, but was not revealed until 1997 due to its top-secret classification.
  - Block cipher (block length and key length can vary) with 1K to 2K bit keys for safety.
  - Based on the practical difficulty of factoring a very large number, i.e. solving the following problem: Given a large  $n = pq$  with  $p$  and  $q$  being prime, find  $p$  &  $q$
- Can be used both for encryption and digital signature.

# Trapdoor One-Way Function

- We know a one-way function is easy to compute, but whose inverse is intractable
- We want more for public-key encryption though - we need to be able to compute the inverse of the one-way function if we have the private key
- So a **trapdoor one-way function** is a one-way function  $f$ , together with a secret  $y$ , such that, given  $f(x)$  and  $y$ , it is easy to compute  $x$
- *The private key here is our trapdoor.*

# Relatively Prime Numbers

- **Definition**

- Relative primes are two numbers that do not have a common divisor.

- **Example:**

- 10 and 11 are relative primes
  - 3 and 16 are relative primes
  - 9 and 5 are relative primes

# Euler's Totient Function

- This function gives the number of positive integers less than  $n$  and relatively prime to  $n$ .
  - By convention  $\phi(1) = 1$
  - If  $p$  is a prime number  $\phi(p) = p-1$
  - If  $n = pq$  where  $p$  and  $q$  are prime and are not equal, then
$$\phi(n) = \phi(pq) = \phi(p) \times \phi(q) = (p-1) \times (q-1)$$
- If  $p$  &  $q$  are equal, then  $\phi(n) = p(q-1)$

## Example:

$$\phi(21) = \phi(3) \times \phi(7) = 12$$

then integers are

1,2,4,5,8,10,11,13,16,17,19,20

Number of integers less than  $n$  and relatively prime to  $n$  is 12



# Euler's Totient Function

## Euler's Theorem

- By convention  $\phi(1) = 1$
- If  $p$  is a prime number  $\phi(p) = p-1$
- If  $n = pq$  where  $p$  and  $q$  are prime and are not equal, then

$$\phi(n) = \phi(pq) = \phi(p) \times \phi(q) = (p-1) \times (q-1)$$

- If  $p$  &  $q$  are equal, then  $\phi(n) = p(q-1)$

$n$	$\phi(n)$
1	1
2	1
3	2
4	2
5	4
6	2
7	6
8	4
9	6
10	4

$n$	$\phi(n)$
11	10
12	4
13	12
14	6
15	8
16	8
17	16
18	6
19	18
20	8

$n$	$\phi(n)$
21	12
22	10
23	22
24	8
25	20
26	12
27	18
28	12
29	28
30	8

# The congruent module of $n$

- A congruence relation (or simply congruence) is an equivalence relation on an algebraic structure.

- Basic Example:

If  $a$  and  $b$  satisfy  $a \bmod n = b \bmod n$ , then  $a \equiv b \bmod n$

e.g.  $73 \bmod 23 = 4$  and  $4 \bmod 23 = 4$

then  $73 \equiv 4 \bmod 23$

# Euler's theorem (aka Fermat–Euler theorem or Euler's totient theorem)

- A generalisation of Fermat's Theorem
- Euler's Theorem states that for every  $a$  and  $n$  that are relatively prime:

$$a^{\phi(n)} \equiv 1 \pmod{n} \quad \text{For any } a, n \text{ where } \gcd(a, n) = 1$$

- E.g.

$$\begin{aligned} a=3; n=10; \phi(10)=4; \\ \text{hence } 3^4 = 81 \equiv 1 \pmod{10} \end{aligned}$$

$$\begin{aligned} a=2; n=11; \phi(11)=10; \\ \text{hence } 2^{10} = 1024 \equiv 1 \pmod{11} \end{aligned}$$

- also have:  $a^{\phi(n)+1} \equiv a \pmod{n}$

# RSA algorithm

- RSA is a block cipher in which the plaintext and ciphertext are integers between 0 and  $n-1$ .
- So the first step is to represent the plaintext block in terms of an integer between **0** and  **$n-1$** .

# RSA algorithm

- Encryption
  - $c = m^e \bmod(n)$
  - *Public-key*  $k_u = \{e, n\}$
- Decryption
  - $m = c^d \bmod(n) = (m^e)^d \bmod n = m^{ed} \bmod n$
  - *Private-key*  $k_r = \{d, n\}$
- Both sender and receiver must know  $n$  &  $e$ .  
Only the receiver knows  $d$
- *Uses two theorems. Fermat's Little Theorem and Euler's Theorem.*

# RSA algorithm

- The algorithm satisfies:
  1. It is possible to find  $e$ ,  $d$  and  $n$  such that  $m^{ed} = m \bmod n$  for all  $m < n$ .
  2. It is easy to calculate  $m^e$  and  $c$  for all  $m < n$
  3. For large values of  $e$  and  $n$  it is not feasible to determine  $d$  given  $e$  and  $n$ .

# RSA algorithm

- Key generation

Determine  $d$  for:  $de \equiv 1 \pmod{\phi(n)}$   
and  $d < \phi(n)$

Select  $p, q$   
 $n = p \times q$   
 $\phi(n) = (p - 1)(q - 1)$

$p$  and  $q$  both prime

Select integer  $e$

$\gcd(\phi(n), e) = 1;$   
 $1 < e < \phi(n)$

$d$

$d \equiv e^{-1} \pmod{\phi(n)}$

Public Key

$k_u = \{e, n\}$

Private Key

$k_r = \{d, n\}$

- Notice:

- $\gcd(\phi(n), e) = 1$  and that  $\phi(n)$  and  $e$  are relative primes.
- To get  $d$ , evaluate  $de \equiv 1 \pmod{\phi(n)}$

# RSA algorithm

- Key generation

Select p,q	$p = 47$ and $q = 59$
$n = p \times q$	$n = 2773$
$\phi(n) = (p - 1)(q - 1)$	$\phi(n) = 2668$
Select integer $e$	$e = 17$
$d$	$d \equiv e^{-1} \text{ mod } \phi(n) \rightarrow d = 157$
Public Key	$k_u = \{17, 2773\}$
Private Key	$k_r = \{157, 2773\}$

See Next  
Slide

Note:  $1 \text{ mod } (\phi(n)) = K \cdot \phi(n) + 1$

$d \equiv e^{-1} \text{ mod } \phi(n) \rightarrow ed \text{ mod } \phi(n) = 1 \text{ mod } \phi(n) = (K \cdot \phi(n) + 1) \text{ mod } \phi(n)$

The choice is limited by the ones which are divisible by  $e$ .



# RSA – Key generation

- $d \equiv e^{-1} \pmod{\phi(n)} \rightarrow d = 157$ . How?

$$d \equiv e^{-1} \pmod{\phi(n)} \rightarrow ed \pmod{\phi(n)} = 1 \pmod{\phi(n)}$$

and  $1 \pmod{\phi(n)} = K \cdot \phi(n) + 1 \pmod{\phi(n)}$

where  $K$  is an integer  $\{1, 2, 3, \dots\}$

Candidates of  $1 \pmod{\phi(n)}$  :

2669	5337	8005	10673	13341	16009	18677	21345	24013	26681
29349	32017	34685	37353	40021	42689	45357	48025	50693	53361
56029	58697	61365	64033	66701	69369	72037	74705	77373	80041

- You will need to find two numbers  $e$  and  $d$  whose product is a number equal to  $1 \pmod{\phi(n)}$ . i.e.  $ed \pmod{\phi(n)} = 1$
- Here are few answers:
  - $2669 = 17 * 157 \rightarrow d = 157$ .
  - $48025 = 17 * 2825 \rightarrow d = 2825$ .
- The above values satisfy the rules:
  - $ed \pmod{\phi(n)} = 1$
  - Both  $e$  and  $d$  are relative primes to  $\phi(n)$ ,

# RSA algorithm

- **Encryption**

Plaintext	$m < n$
Ciphertext	$c = m^e \bmod n$

- **Decryption**

Ciphertext	$c$
Plaintext	$m = c^d \bmod n$

# RSA Example

- Encode the literal characters (space), a, b, ..., z with the biagrams 00, 01, ..., 26
- The message is encoded in block of length two

E	R	R	A	R	E
05	18	18	01	18	05
0518		1801		1805	

- Encryption, e.g  $c_1 = 518^{17} \bmod 2773 = 1787$

1787	2003	2423
------	------	------

- Decryption, e.g  $m_2 = 2003^{157} \bmod 2773 = 1801$

0518	1801	1805
------	------	------

# Why RSA Works

- Suppose  $(n,e)$  and  $(n,d)$  are key pair and  $m < n$ .

Then  $\mathbf{m} = c^d \bmod(n) = (m^e)^d \bmod n = \mathbf{m^{ed} \bmod n}$

- To prove
$$\begin{aligned}(m^e)^d &\equiv m \bmod n \\ m^{ed} &= m^{(1+k(p-1)(q-1))} \\ &= m(m^{(p-1)(q-1)})^k \\ &\equiv m(1^k) \bmod n \\ &\equiv m \bmod n\end{aligned}$$

But  $0 < m < n$ , so congruence mod  $n$  implies genuine equality.

# RSA Security

- **Possible approaches to attacking RSA are:**
  - Brute force key search - infeasible given size of numbers
  - Mathematical attacks - based on difficulty of computing  $\phi(n)$ , by factoring modulus  $n$
  - Timing attacks - on running of decryption
  - Chosen ciphertext attacks - given properties of RSA

# Problem with Public-key Encryption

- Need relatively **long keys**, 1024 or more bits
- **Computational costs**: slower than DES by a factor of a thousand!
- **Long-plaintext security issues**
  - For longer plaintext → split into “blocks” and then encrypt these separately.
  - Essentially encrypting these blocks using the public-key equivalent of ECB mode for a block cipher, which is not desirable.
  - However, there are no alternative modes of operation proposed for public-key encryption.

# Problem with Public-key Encryption

- Symmetric cryptography and Public-key cryptography have both strength and weakness.
- In particular, we often need the good properties of both:
  - Ease of key sharing from Public-key cryptography.
  - Efficiency and simplicity from Symmetric cryptography.
- *So what should we do?*




# Hybrid Encryption

For secure communications with potentially high traffic, the **hybrid cryptography** gives an ideal answer.



# Hybrid Encryption

- Alice wishes to exchange several messages with Bob securely. We assume Alice and Bob share their public keys.
- First, Alice creates a fresh session key (symmetric encryption key),  $K_{AB}$ .
  1.  $A \rightarrow B : \{K_{AB}\}_{K_{B, pub}}$
  2.  $B \rightarrow A : \{M1\}_{K_{AB}}$
  3.  $A \rightarrow B : \{M2\}_{K_{AB}}$
  4.  $B \rightarrow A : \{M3\}_{K_{AB}}$
- (In Step 2, we assume Bob decrypts the message and obtains  $K_{AB}$ .)

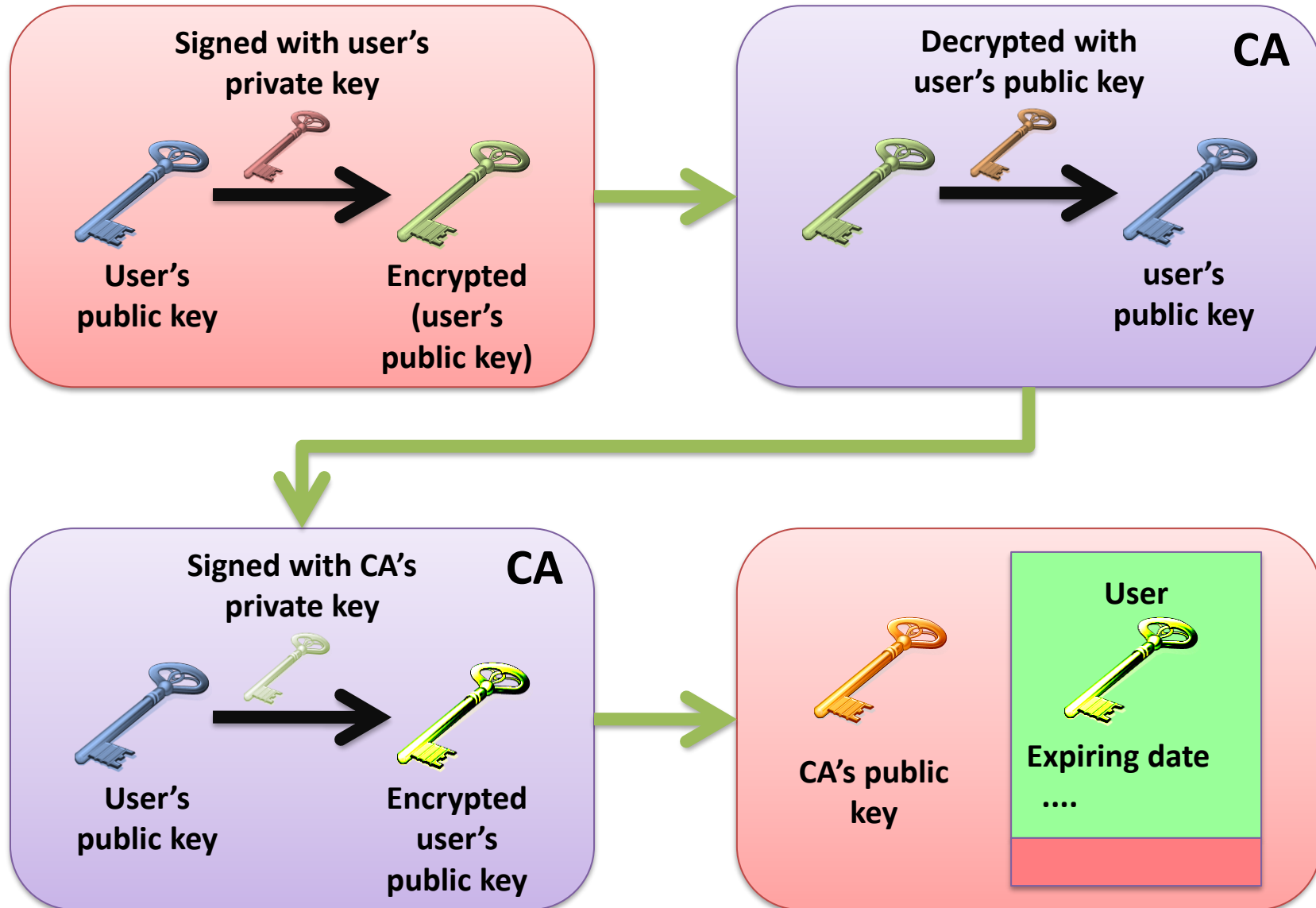


We can see that Public Keys are useful, but the authenticity of public keys is still vital ...

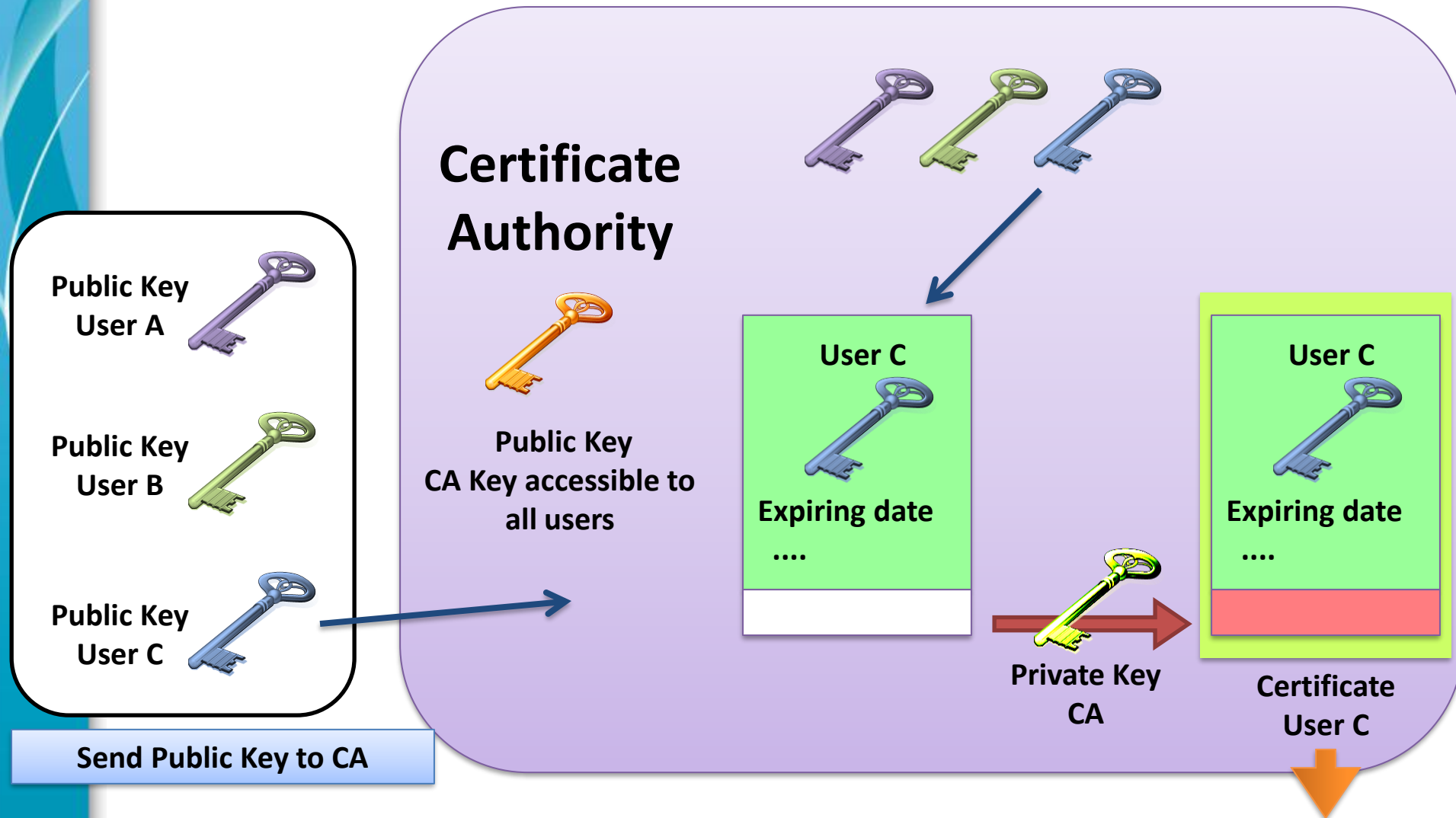
# Certificates (introduction)

- Alice generates a key pair and signs the public key and identification information with the private key. Sends the encrypted information to the certification authority.
- The certification authority verifies Alice's signature on the key and the ID information (Optional: CA verifies Alice's ID through other methods; email, credit card, etc.)
- CA signs the public key and ID with the CA private key, creating a certificate.
- Alice verifies the key, ID and CA signature.
- Alice and/or CA publish the certificate.

# Obtaining a Certificate



# Obtaining a Certificate

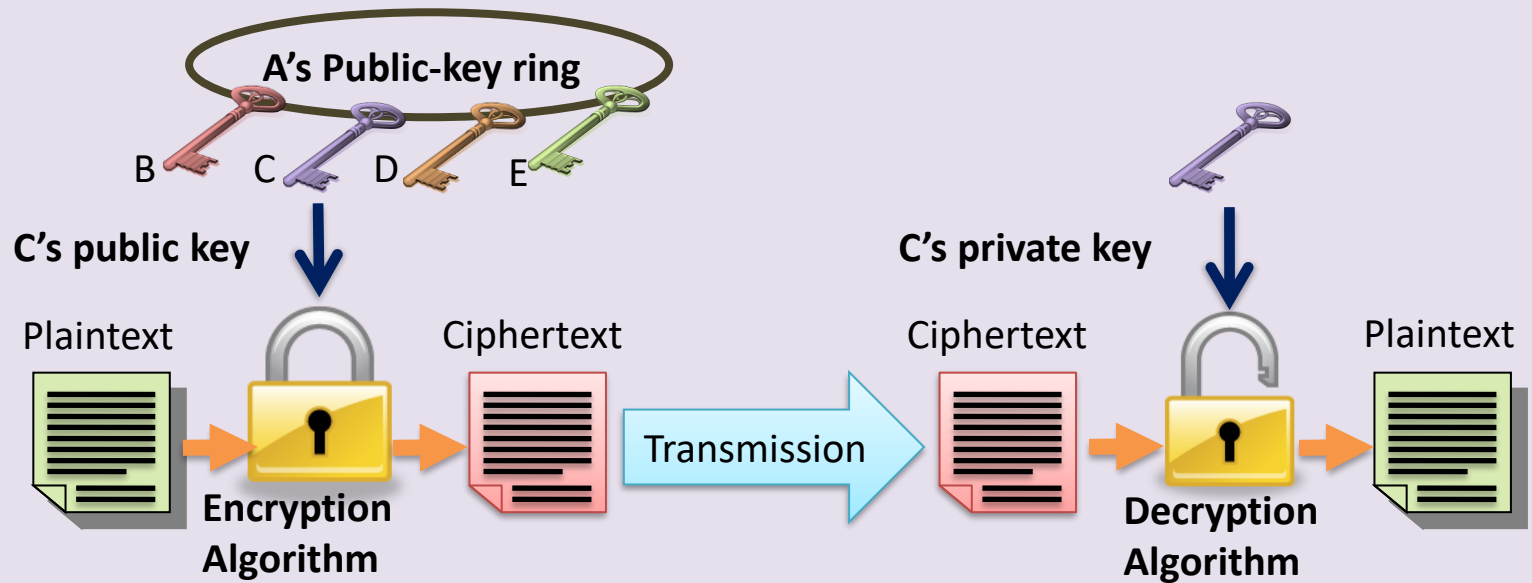


# Obtaining a Certificate

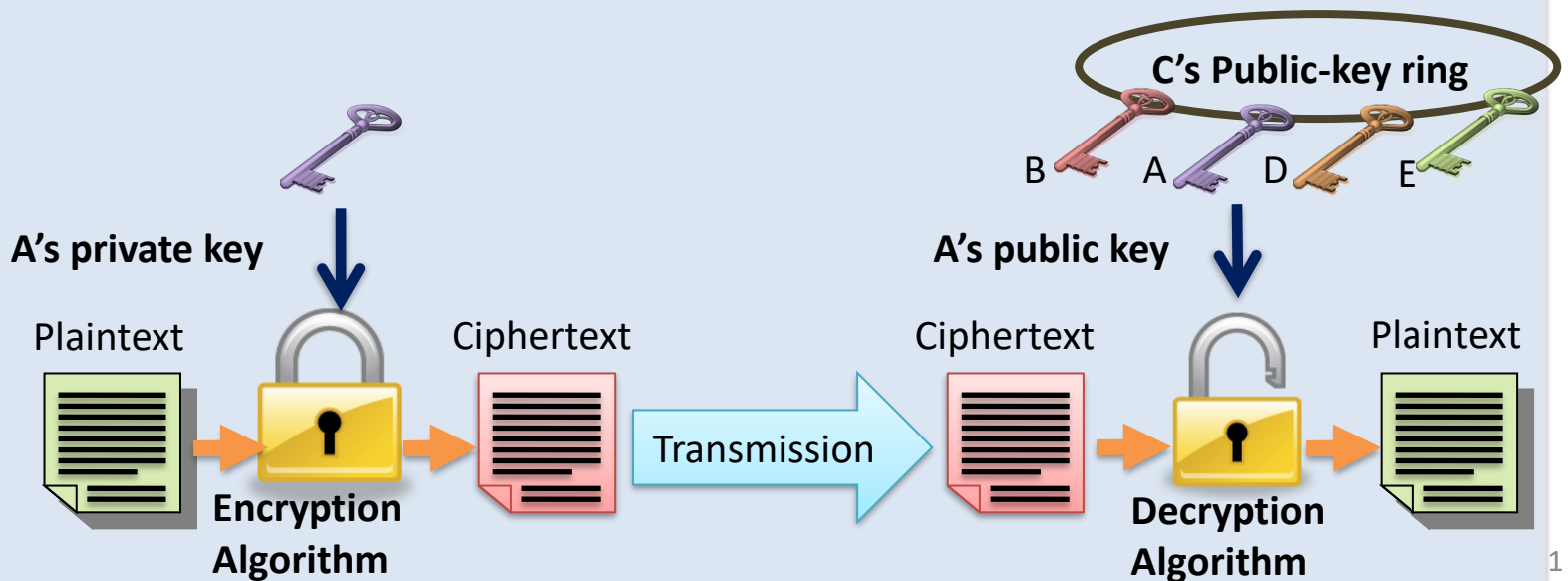
- A certification authority guarantees the connection between key and any user with access to CA can get any certificate from it:
  - A person
  - A role (“Director of Personnel”)
  - An organisation
  - A piece of hardware/software
  - An account
- Only the CA can modify a certificate.
- Because cannot be forged, certificates can be placed in a public directory.

# Public Key Cryptography Summary

## Encryption



## Authentication



# Message Authentication (Data integrity)

## Prevent:

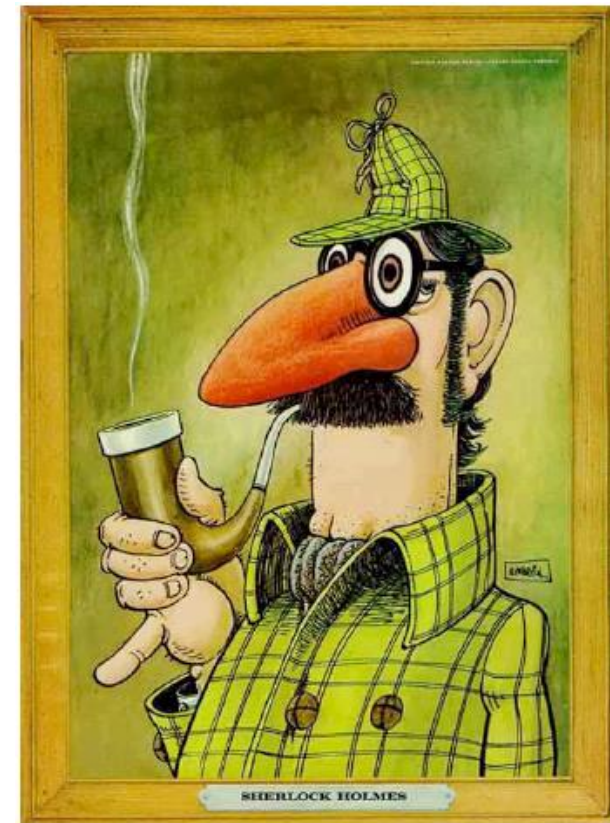
- Content modification (message integrity)
- Masquerade (validation of the originator's identity)
- Sequence modification
- Time modification
- Repudiation

Capture and replay of valid  
authentication sequence  
(pretending being someone else)



# Authentication

- Prove or show to be true or genuine
  - Contents of message
  - A person, a computer or an address
  - A timestamp
  - A signature

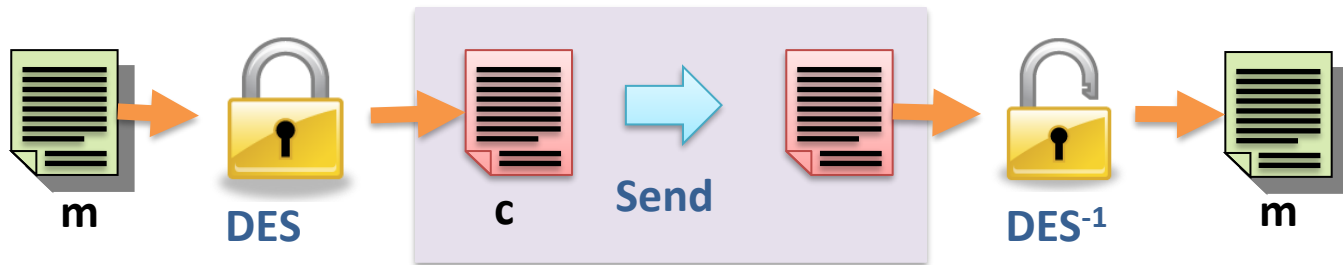


# Data Integrity - Introduction

- Data Integrity: assurance that data has not been altered in an *unauthorised* (which includes accidental) manner.
- Data integrity is not concerned with the *prevention* of alteration of data, but provides a means for detecting whether data has been manipulated in an unauthorised way.
- Can encryption provide data integrity?

# Message Authentication

- Data integrity can be done by using conventional encryption.



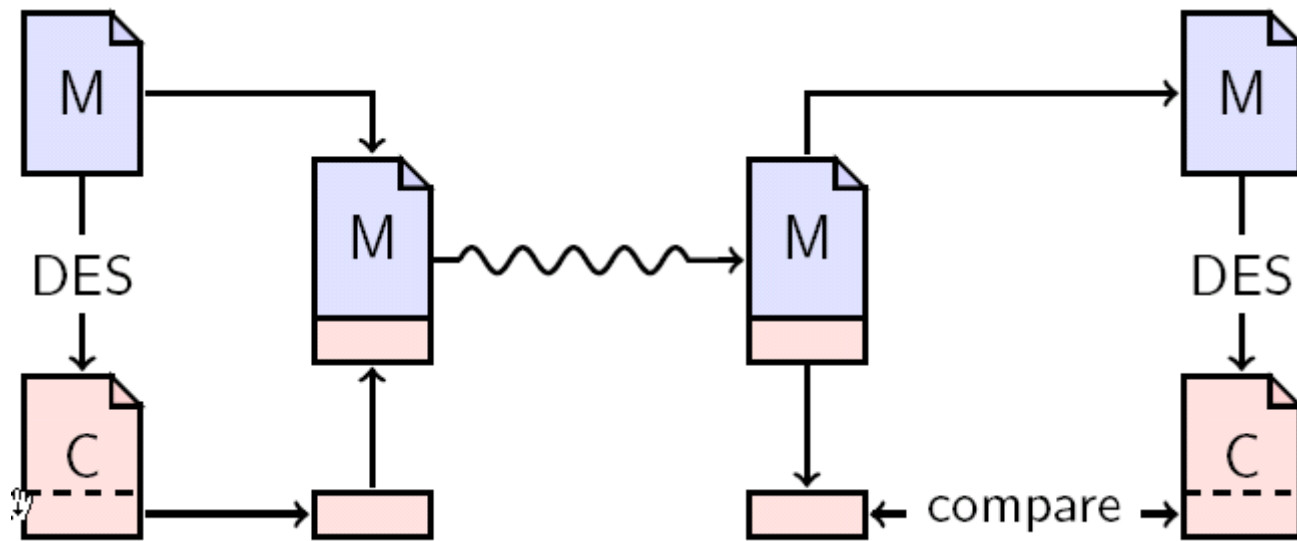
- If only the sender and receiver share the key.
- If the message includes error-detection code and a sequence number the receiver is assured that the message has not been altered and is in the correct sequence.
- If the message includes a time-stamp the receiver is assured that the message has not been delayed.
- But it does not provide a signature. i.e. The receiver can fabricate a message, encrypted using the share key and attribute it to the sender.

# Message Authentication

- It is not always desirable to encrypt a message to provide authentication
  1. **Broadcasting a message** to many destinations with an authentication tag;
    - It is cheaper and more reliable to have one destination responsible for monitoring of authenticity.
    - If the message is not authentic then raise an alarm.
  2. Due to heavy load in the system, **decryption cannot be afforded**, so it is better to check the authentication of the message in a random basis.
  3. **Authentication of a computer program.** The program can be checked when assurance about its *integrity* is needed.

# Authentication using conventional encryption

- Consider the following authentication method



- Suppose the ECB mode is used for the DES encryption, is it a good method for message authentication?

# A simple example for Data integrity

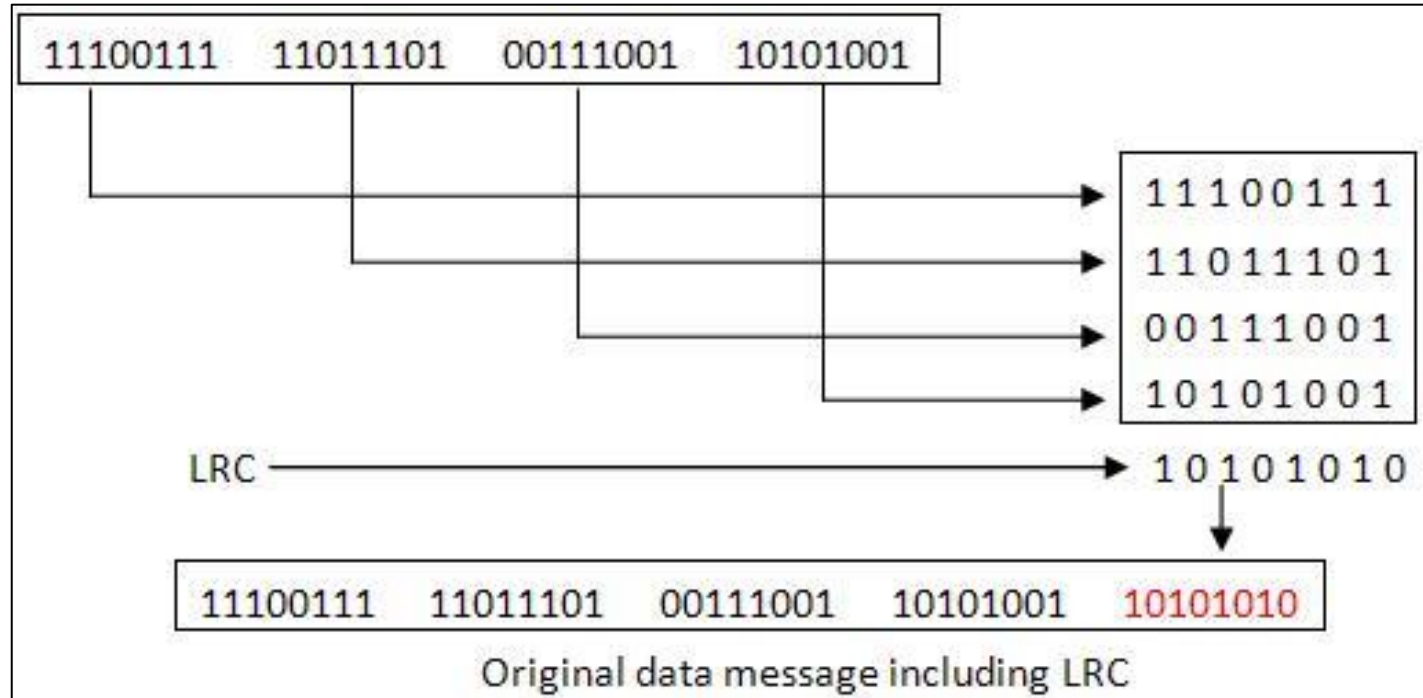
- Longitudinal redundancy check (LRC)
  - The input is a sequence of  $n$ -bit blocks
  - The input is processed one block at a time, to produce an  $n$ -bit hash function

$$c_i = b_{i1} \oplus b_{i2} \oplus \dots \oplus b_{im}$$

- $b_{i,j}$  is the  $i$ th bit of the  $j$ th block

	Bit 1	Bit 2	...	Bit $n$
Block 1	$b_{11}$	$b_{21}$	...	$b_{n1}$
Block 2	$b_{12}$	$b_{22}$	...	$b_{n2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Block $m$	$b_{1m}$	$b_{2m}$	...	$b_{nm}$
Hash Code	$c_1$	$c_2$	...	$c_n$

# A simple example for Data integrity



- Good for random data, not so good for predictable formatted data.

# Hash functions

- Cryptographic hash functions
- A hash function accepts a *variable-size* message  $m$  and produces a *fixed-size* **message digest**  $h(m)$  as output.
- A hash function does **not** use a secret key.
- The sender sends the message and message digest so the receiver can authenticate the message.
- Hash functions are publicly computable. We always assume an attacker knows the details of a hash function.



# Practical properties of a hash function

1.  *$h$  can be applied to a block of data of any size*
2.  $h$  produces a fixed-length output.
3.  $h(m)$  is easy to compute for any  $m$ .

# Security Properties of a hash function

1. Given a code  $c$  it is computationally infeasible to find  $m$  such that  $h(m) = c$
2. Given  $m$  it is computationally infeasible to find  $n \neq m$  such that  $h(m) = h(n)$
3. It is computationally infeasible to find any pair  $(m,n)$  such that  $h(m) = h(n)$

# Security Properties of a hash function

## 1. Preimage Resistance

Given an output (hash) value  $c$ , it should be a difficult operation to find any input value  $m$  such that  $h(m)=c$ .

Output  $h(m)$  of a function  $h$  is often called the **image of  $m$** , and hence  $m$  is referred to as a **preimage of  $h(m)$** .

## 2. Second Preimage Resistance

given an input  $m$  and its output (hash) value  $h(m)$ , it should be a difficult operation to find any other input value  $n$  such that  $h(m)=h(n)$ .

## 3. Collision Resistance

for a hash function  $h$ , it is hard to find two different inputs  $m$  and  $n$  such that  $h(m)=h(n)$ .

# Application of Hash functions

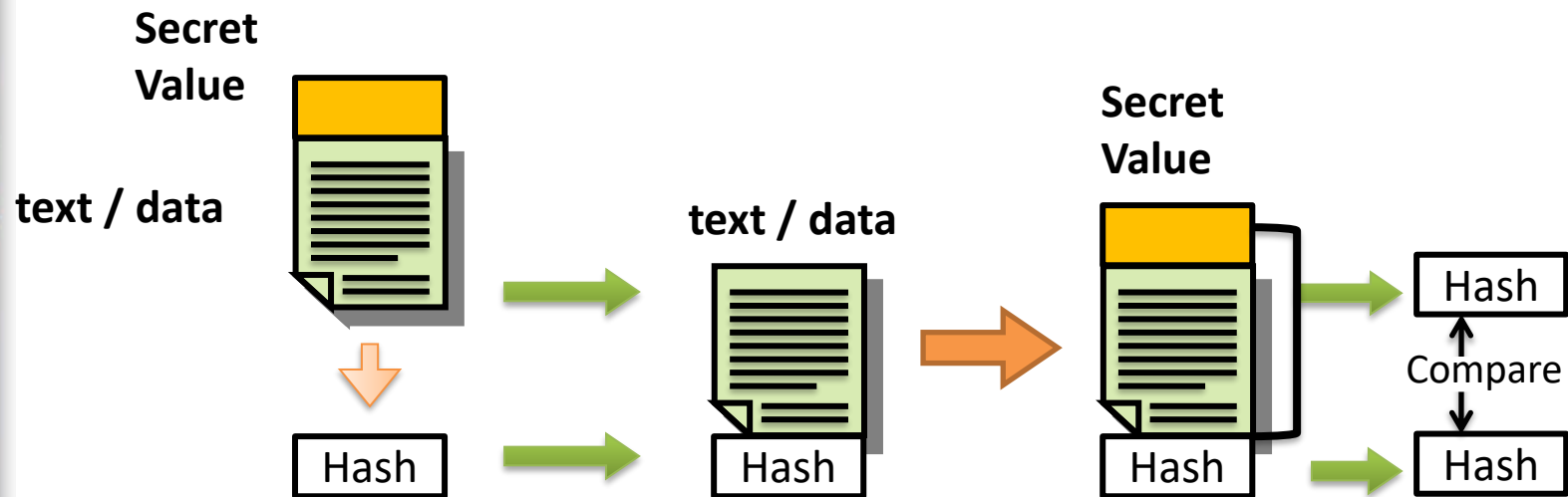
- **Application Requiring Preimage Resistance**  
password storage protection
- **Application Requiring Second Preimage Resistance**  
Software downloading with a hash of the executable code
- **Application Requiring Collision Resistance**  
Bob writes an IOU message, sends it to Alice, and she signs it. Bob finds two messages with the same hash, one of which requires Alice to pay a small amount and one that requires a large payment. Alice signs the first message, and Bob is then able to claim that the second message is authentic.

# Hash functions

- It is possible to use a hash without using encryption.
- The sender and receiver share a common secret value  $S_{AB}$  value and the message.
- The hash is obtained from the concatenation of the secret value and the message.
- The sender sends the message and the hash (not the secret value).

# Hash functions

- The receiver concatenates the secret value & received message.
- The receiver evaluates the hash function and compares it with the sender's hash function. If equal, the message is authentic.



# Secure Hash Algorithm (SHA)

- SHA originally designed by NIST & NSA in 1993
- Revised in 1995 as SHA-1
- 160-bit hash values
- Later in 2005 results on security of SHA-1 have raised concerns on its use in future applications.
- SHA3 was announced by NIST in October 2012.

# SHA1 (Secure Hash Algorithm 1)

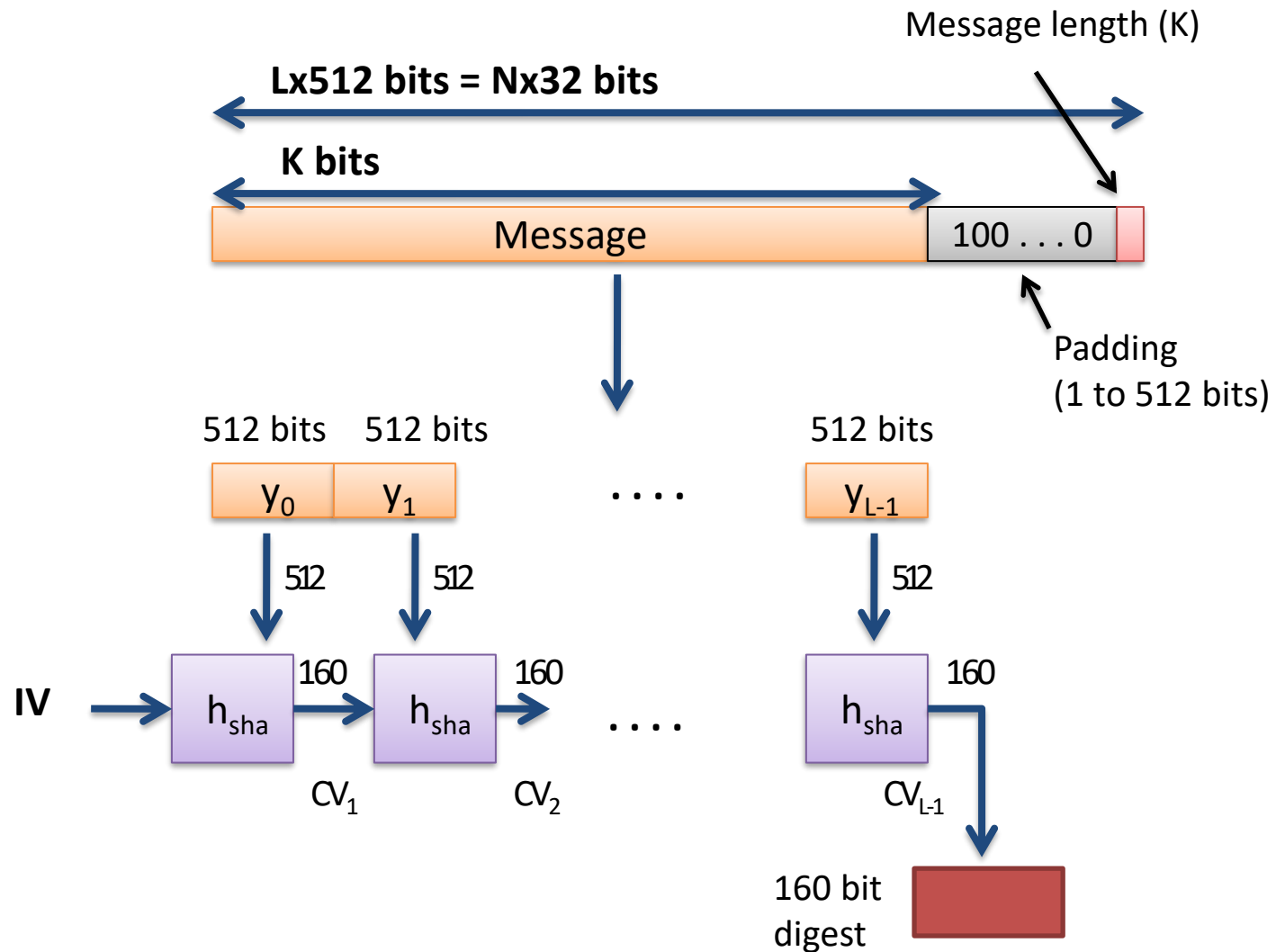
- The processing consists of:
  1. Append padding bits
  2. Append length
  3. Initialise MD buffer
  4. Process message in 512-bit blocks
  5. Output: 160 bits



# Hash

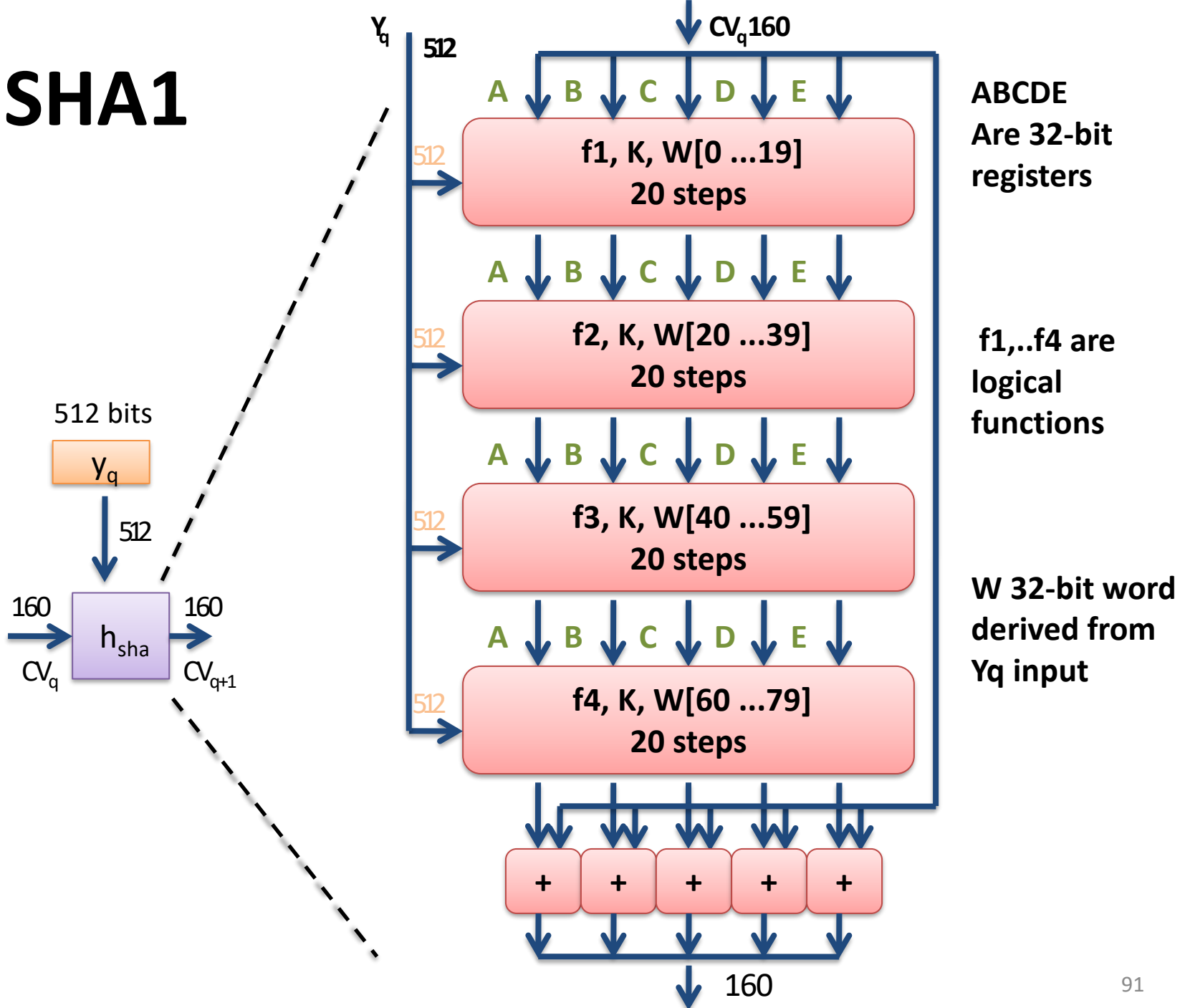
- **The output:**
  - Every bit of hash code is a function of every bit of the input.
  - It is unlikely that two messages chosen at random will have the same hash code.
  - It is infeasible to produce two different messages having the same message digest.

# SHA1

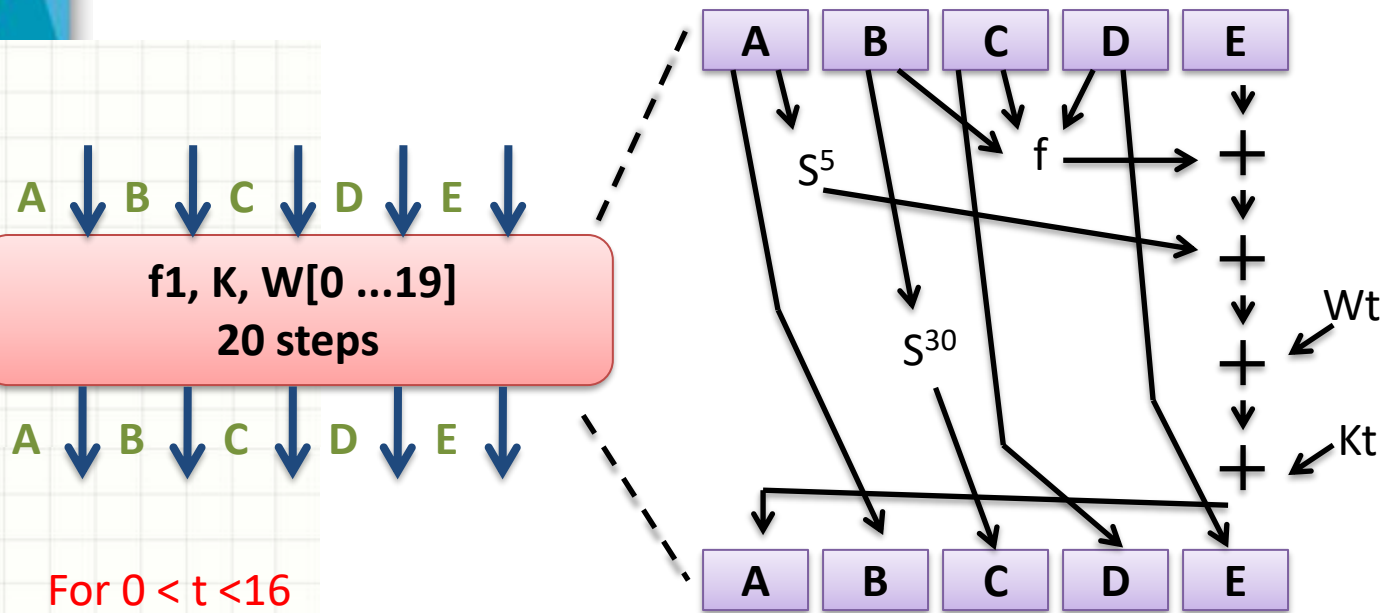


CV: Compression Value.

# SHA1



# SHA1



Example of the  
logical function

$$f1 = (B \wedge C) \vee (\bar{B} \wedge D)$$

$\wedge$  = logical AND

$\vee$  = logical OR

$\bar{\phantom{x}}$  = logical NOT

For  $0 < t < 16$

Divide the  $Y_q$  into 16 words  
 $W_0, W_1, \dots, W_{15}$ , where  $W_0$   
is the left-most word.

For  $t=16$  up to 79,

$K_t$  = additive constant

$W_t$  32-bit word derived from  $Y_q$

$$W_t = S^1(W_{t-16} \text{ XOR } W_{t-14} \text{ XOR } W_{t-8} \text{ XOR } W_{t-3})$$

$S^n$  =  $n$  circular left shift

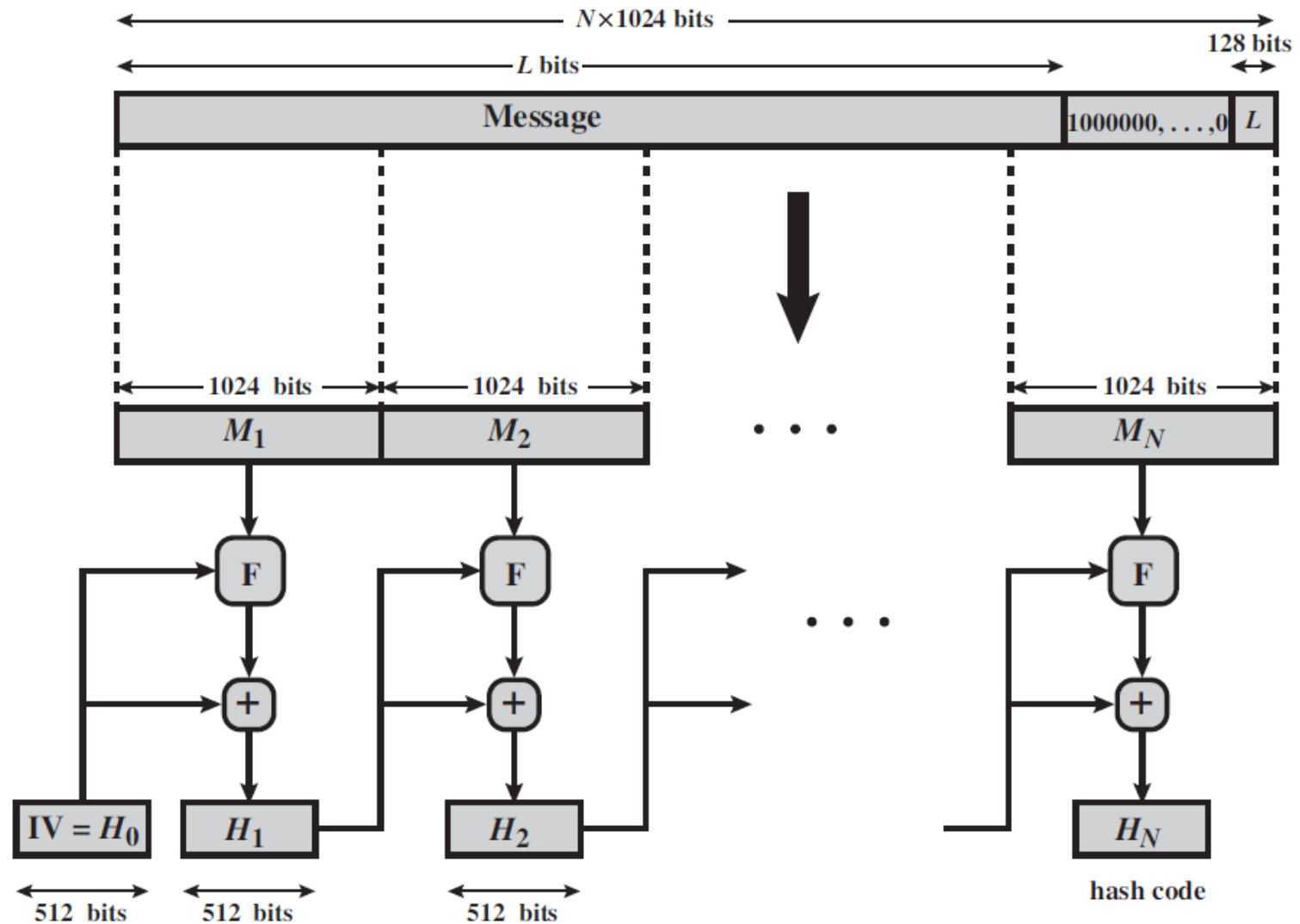
# Revised Secure Hash Standard

- NIST issued revision FIPS 180-2 in 2002
- Adds 3 additional versions of SHA
  - SHA-256, SHA-384, SHA-512
- Designed for compatibility with increased security provided by the AES cipher structure & detail is similar to SHA-1.

# SHA Versions

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message digest size	160	224	256	384	512
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size	512	512	512	1024	1024
Word size	32	32	32	64	64
Number of steps	80	64	64	80	80

# SHA-512 Overview



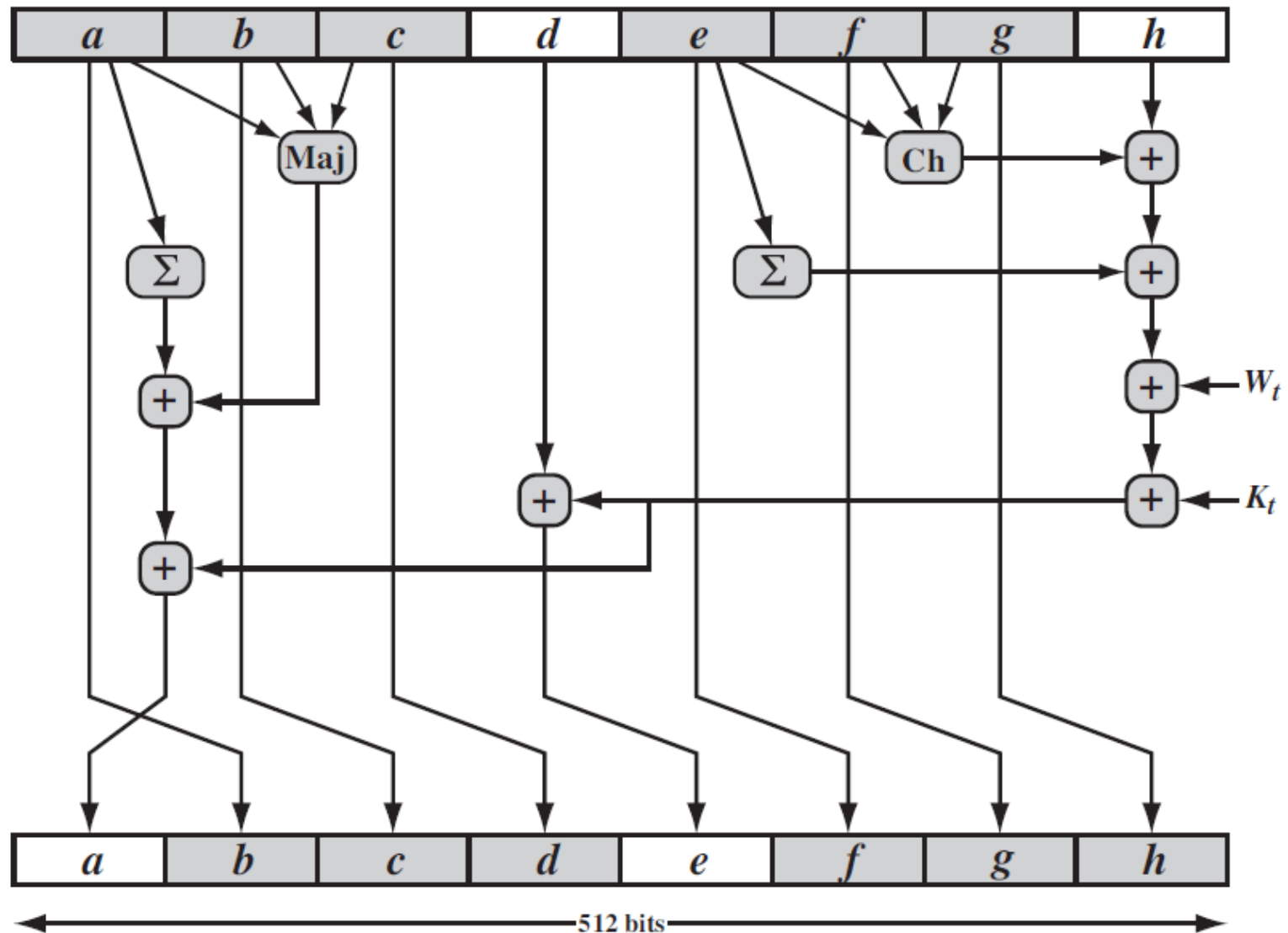
$+$  = word-by-word addition mod  $2^{64}$

# SHA-512 Compression Function

- Processing message in 1024-bit blocks
- Consists of 80 rounds
  - Updating a 512-bit buffer
  - Using a 64-bit value  $W_t$  derived from the current message block
  - and a round constant based on cube root of first 80 prime numbers



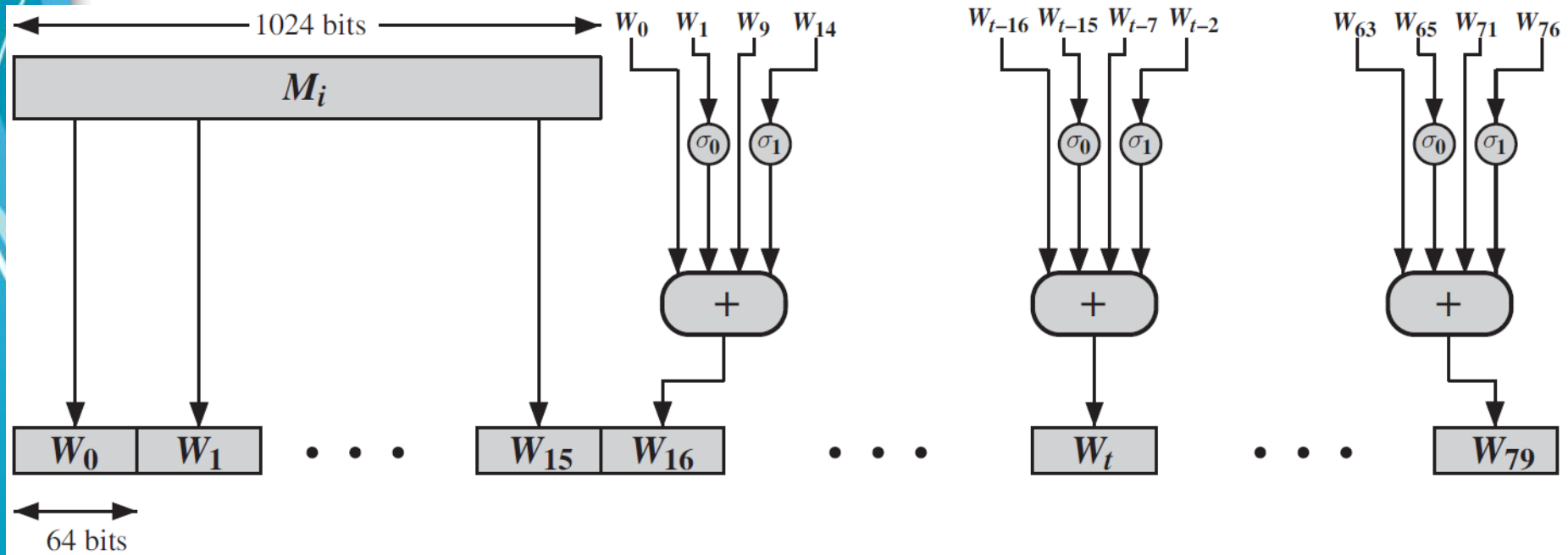
# SHA-512 Round Function



$$\text{Ch}(E, F, G) = (E \wedge F) \oplus (\neg E \wedge G)$$

$$\text{Ma}(A, B, C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$

# SHA-512 Round Function



# SHA-3

- SHA-1 not yet "broken"
  - Similar to broken MD5 & SHA-0
  - Hence, considered insecure
- SHA-2 (esp. SHA-512) seems secure
  - Same structure and mathematical operations as predecessors so have concern.
- NIST announced in 2007 a competition for the SHA-3 next gen NIST hash function
  - The winning design for SHA3 was announced by NIST in October 2012.
  - Cryptographic hash function to complement SHA-2.

# SHA-3 Requirements

- Replace SHA-2 with SHA-3 in any use
  - Same hash sizes
- Preserve the online nature of SHA-2
  - Must process small blocks (512 / 1024 bits)
- Evaluation criteria
  - Security close to theoretical max for hash sizes
  - Cost in time & memory
  - Characteristics: such as flexibility & simplicity

# Other hashes

	MD5	SHA-1	SHA-2	SHA-3	RIPEMD-160
<b>Digest length</b>	128 bits	160 bits	512 bits	512 bits	160 bits
<b>Basic Unit of processing</b>	512 bits	512 bits	1024 bits	576 bits	512 bits
<b>Number of steps</b>	64	80	80	24	160
<b>Maximum message length</b>	$\infty$ (4 rounds of 20)	$2^{64} - 1$ (4 rounds of 20)	$2^{128} - 1$ (4 rounds of 20)	Unlimited	$\infty$ (5 rounds of 16)

# Other Hash Function Uses

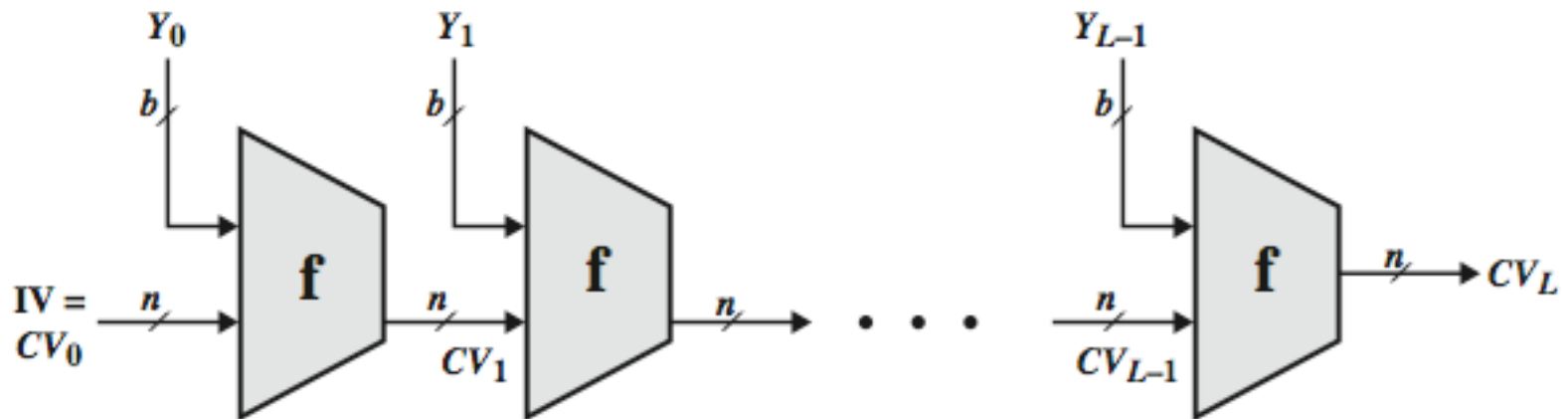
- To create a one-way password file
  - Store hash of password not actual password
- For intrusion detection and virus detection
  - Keep & check hash of files on system
- Pseudorandom function (PRF) or pseudorandom number generator (PRNG)

# Attacks on Hash Functions

- Both brute-force attacks and cryptanalysis
- A preimage or second preimage attack
  - Find  $y$  such that  $H(y)$  equals a given hash value
  - For an  $m$ -bit hash value, the adversary would have to try on average  $2^{m-1}$  values of  $y$  to find one that generates a given hash value  $h$ .
- Collision resistance attack
  - Find two messages  $x$  &  $y$  with same hash so  $H(x) = H(y)$
  - Birthday Attack: value  $2^{m/2}$  determines strength of hash code against brute-force attacks
  - 128-bits inadequate, 160-bits suspect

# Hash Function Cryptanalysis

- Cryptanalytic attacks exploit some property of the algorithm, so it is faster than exhaustive search
- Hash functions use iterative structure
  - Process message in blocks (incl length)
- Attacks focus on collisions in function **f**





# The birthday paradox

- What is the probability that in this lecture there are two of us which have the same birthday. Let's say that there are approximately 140 in the class
- Consider a small group of, say five.
- The number of ways which five dates can be chosen is

$$365 \times 365 \times 365 \times 365 \times 365$$

- Of all of these ways only

$$365 \times 364 \times 363 \times 362 \times 361$$

are such that no two dates are the same.

# The birthday paradox

- The probability that five dates chosen at random is

$$\frac{(365 \times 364 \times 363 \times 362 \times 361)}{365^5} = 0.97$$

- Hence the probability that in a group of five persons two of them have the same birthday is 3%.
- Doing the same procedure with 140 persons the probability of two people having the same birthday is 99.9999999999%

# The Birthday Attack

- A related question is, how big does a group of people have to be to have a **50%** chance that there are two people with the same birthday? Following the above approach we have that the group should have at least **23 people**.

# The Birthday Attack

- The birthday attack is based on the birthday paradox
  - Number of days per year  $\leftrightarrow$  number of possible hashes
  - How many pairs of modified original/fraudulent messages we need to create to have a 50% chance that they have the same hash.
  - For an  $m$ -bit hash we need  $2^{m/2}$  modified messages.

# The Birthday Attack

- An attacker needs to find a message  $M'$  such that the hash of this message is the same as the hash of the original message  $M$ , that is  $H(M') = H(M)$ .
- If the hash is a 64-bit hash then using brute force, on average, the attacker needs to try about  $2^{63}$  messages to find one that matches the hash code of the original message.

$$2^{63} = 9,223,372,036,854,775,808$$

# The Birthday Attack

- It is based on the birthday paradox and here how it works:
  - Alice is ready to 'sign' a message by encrypting it, using her private-key, to generate the  $m$ -bit hash code of the message.
  - Eve generates  $2^{m/2}$  variations of the message, all of which convey the same meaning.
  - Eve also generates an equal number of messages all of which are variations of the fraudulent message that Eve wants to substitute for the original one.
  - Eve searches in the two sets of messages for a pair that produce the same hash code. i.e. [ hash(variation of original) = hash(variation of fraudulent) ]
  - The probability success is greater than 0.5.

# The Birthday Attack

- For the 64-bit hash code, Eve needs to create  $2^{32}$  variations of the message ( $2^{32} = 4,294,967,296$ )
- Eve presents the valid variation of the original message to Alice for signature. After Alice signs the hash, the hash is attached to the fraudulent version of the message and that is it.

# The Birthday Attack

- For Eve it is not difficult to create variations of the message. Here is an example (32 versions of simple message).

*Dear Anthony,*

*This letter is to introduce you to Mr. Alferd P. Borton,*

*I am writing to you - -*

*the new chief ...*

*newly appointed senior*



# Message Authentication Code (MAC)

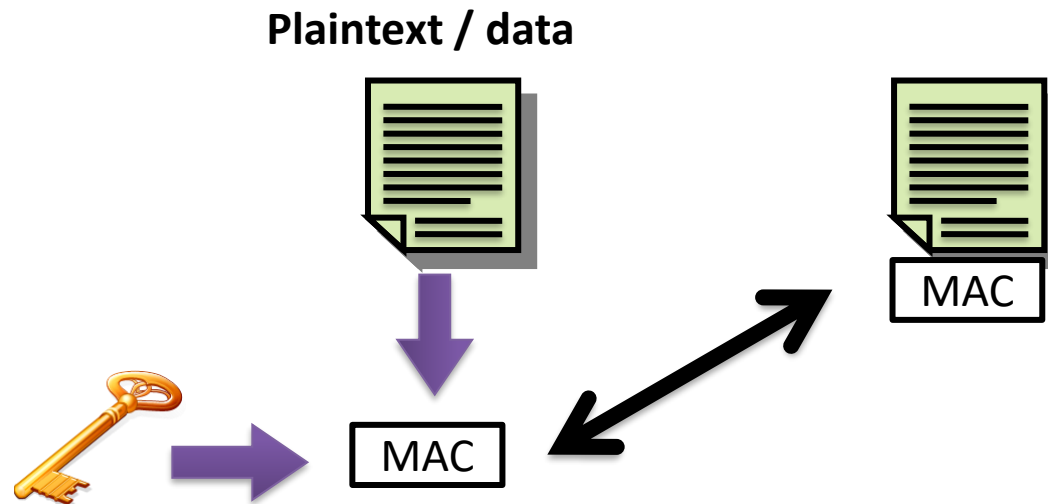
aka keyed hash function

- Message Authentication involves generating a small block of data (the MAC) that is appended to the message.
- The MAC depends on the contents of the message and a secret key  $MAC(m) = f(k_{AB}, m)$ , where  $m$  is the plaintext,  $k_{AB}$  is the shared key between users A and B.
- The message plus MAC are transmitted to the recipient.
- The recipient, which knows the secret key, calculates the MAC of the message and compares it with the MAC just received.
- Any changes in the message are noticed if the MAC calculated by the receiver and the MAC from the sender mismatch.

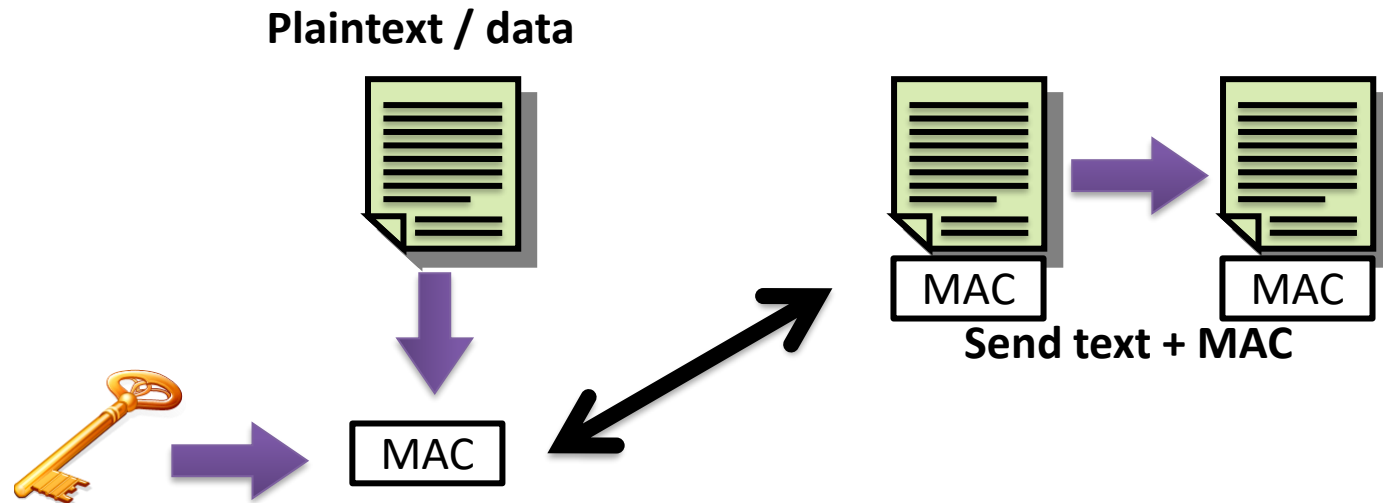
# Authentication

- If the received MAC matches the calculated MAC, then the receiver is assured that the message
  - has not been altered (integrity of the message).
  - is from the alleged sender (originator identity validation).
  - If the message has a sequence number, the receiver can check it has a proper sequence.

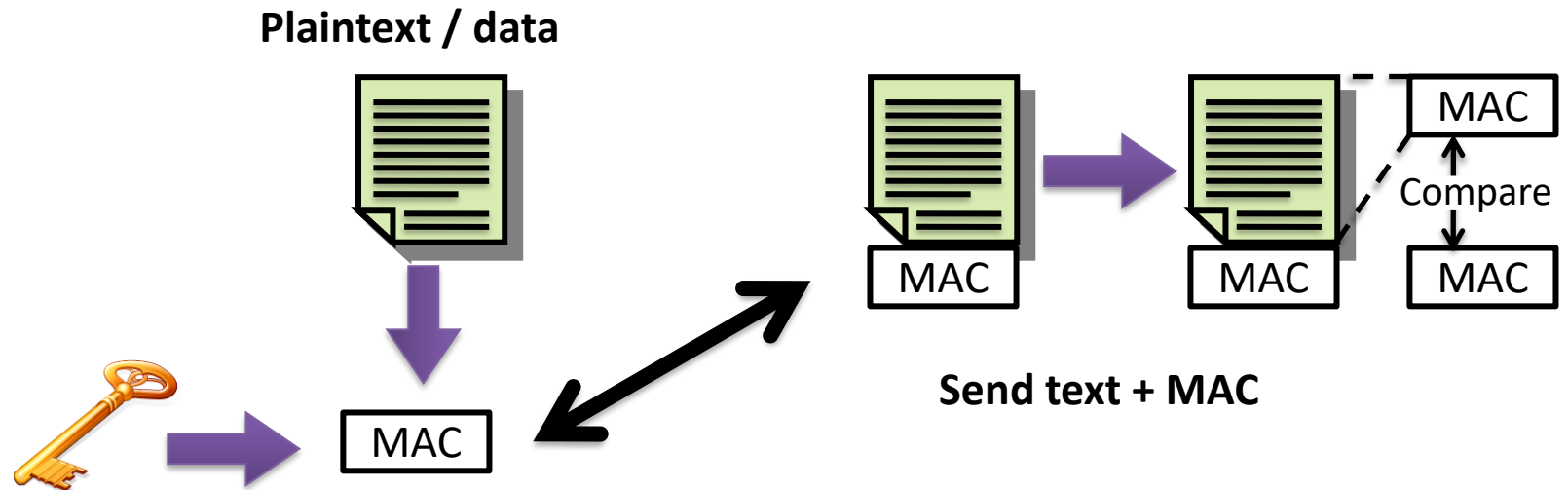
# MAC



# MAC

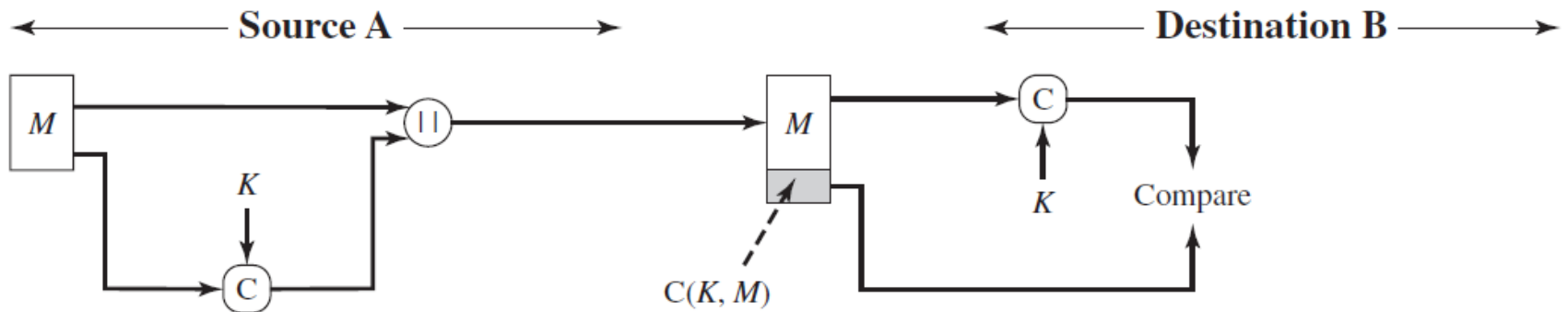


# MAC



# MAC

$$MAC = C(K, M)$$



# Security of MACs

- As with block ciphers, there are:
  - **Brute-force** attacks exploiting
    - Strong collision resistance hash have cost  $2^{m/2}$
    - MACs with known message-MAC pairs
      - Can either attack keyspace (key search) or MAC
      - At least 128-bit MAC is needed for security
  - **Cryptanalytic attacks** exploit structure
    - More variety of MACs so harder to generalise about cryptanalysis

# An authentication standard

- From the National Bureau of Standards (NBS)
  - Use DEA to encrypt the message
  - The MAC is the last bits of Ciphertext (16-32 bits is typical)
- Is similar to encryption, **but** it doesn't need to be reversible (we don't need decryption).



# HMAC

- HMAC is a MAC derived from a cryptographic hash code, that is, it incorporates a secret key into an existing hash algorithm.

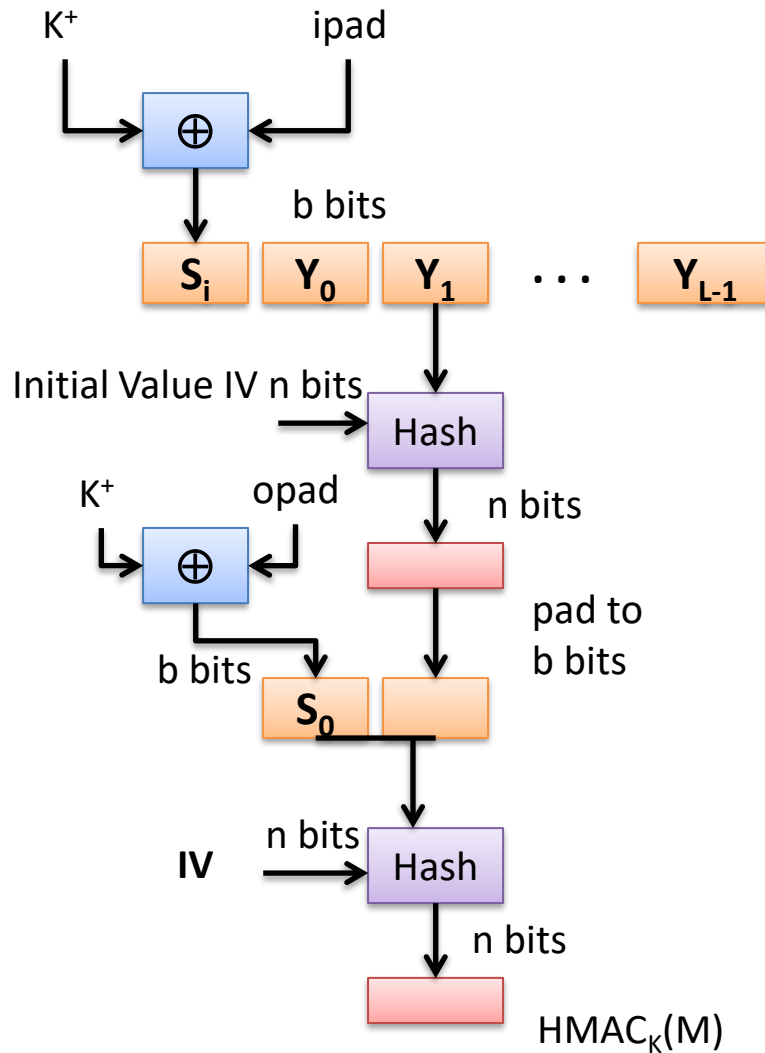
## Why?

1. Cryptographic hash functions execute faster (software) than conventional encryption algorithm (e.g. DES)
2. Code for cryptographic hash functions is widely available.

# HMAC

- Design Objectives
  - To use available hash functions.
  - To allow for easy **replaceability** of the embedded hash function.
  - To preserve the original performance of the hash function.
  - To use and handle keys in a simple way.
  - To have a well-understood cryptographic analysis of the strength of the authentication mechanism.

# HMAC



$K^+$  = secret key, padded with zeros so is  $b$ -bits long.

$Y_i$  =  $i$ -th block of the message  $M$

$L$  = number of blocks in  $M$

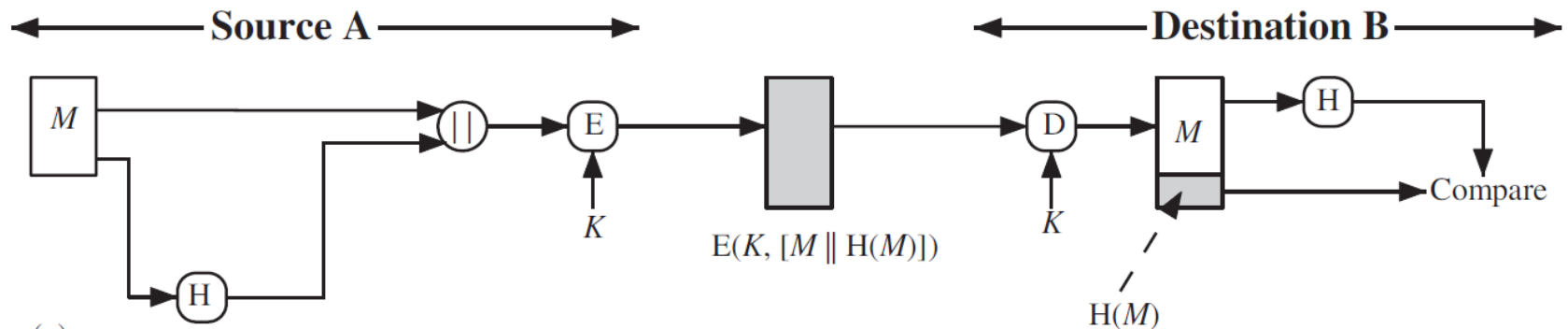
$B$  = number of bits in a block

$n$  = length of the hash code

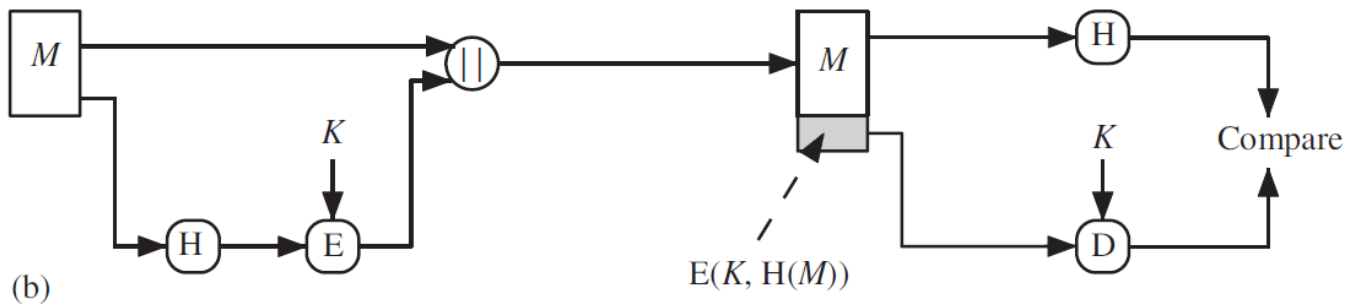
$ipad$  = padding

$opad$  = padding

# Hash Functions & Message Authentication (1)

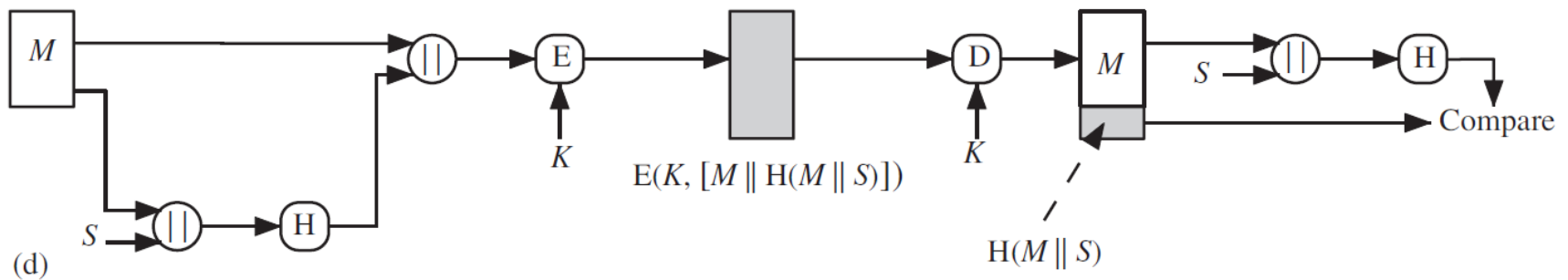
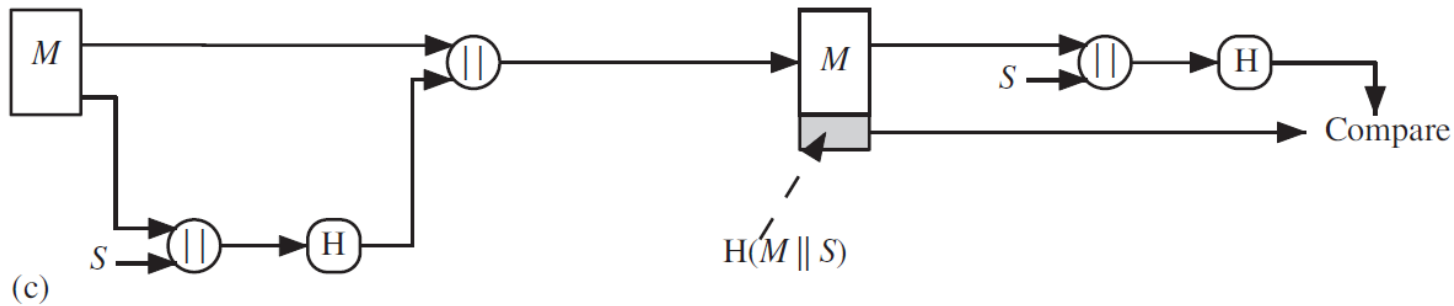


(a)



(b)

# Hash Functions & Message Authentication (2)



# Digital Signature

Digital signature scheme provides:

- **Data origin authentication of the signer.** A digital signature validates the underlying data in the sense that assurance is provided about the integrity of the data and the identity of the signer.
- **Non-repudiation.** A digital signature can be stored by anyone who receives it as evidence.

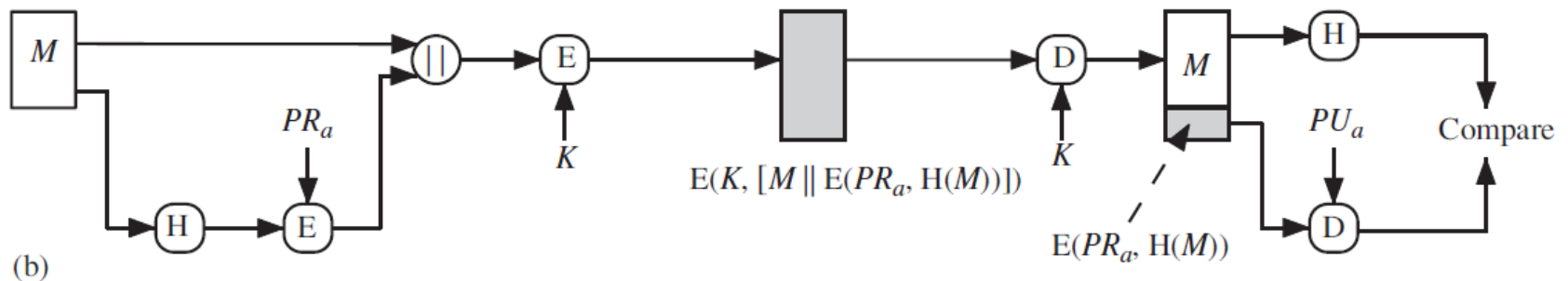
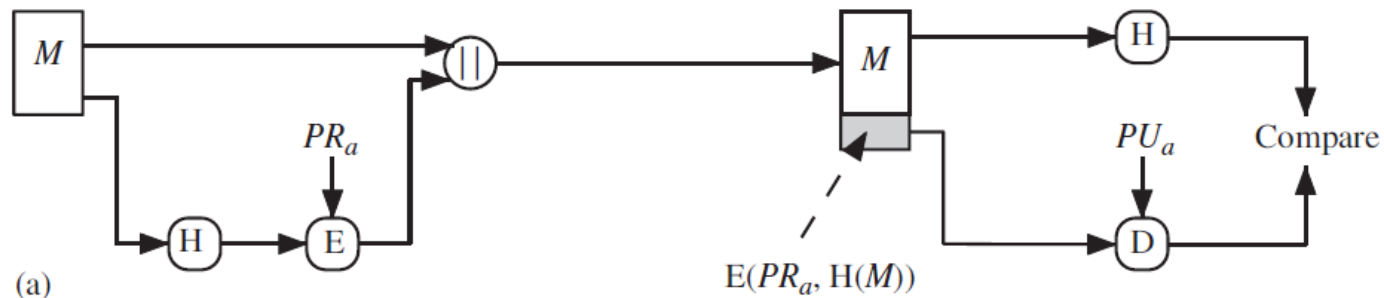
# Digital Signature

A digital signature on some data will need to be computed from:

- **The data.**
- **A secret parameter known only by the signer.**

# Hash Functions & Digital Signatures

← Source A →                      ← Destination B →





# Summary

- Public-Key Encryption
  - Introduction
  - *Diffie-Hellman* Key Exchange
  - RSA algorithm
  - Certificate
  - Data Integrity/Message Authentication
  - Digital signature
  - Attacks
    - Man-in-the-middle, Birthday/hash attacks.

# Next..

- Network Authentication and Access Control Services using Kerberos
- IPSec
- Firewalls

