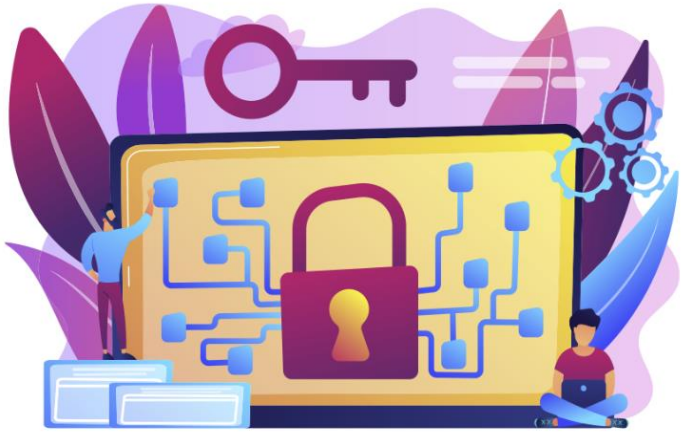
Lecture 2

Modern Cryptography

Dr. Charnon Pattiyanon
Assistant Director of IT, Instructor
Department of Artificial Intelligence and Computer Engineering
CMKL University

This Lecture's Outline

- Upon successful of this lecture, you will know about:
  - **Basic** and **formal definition** of cryptography and its operations.
  - **Early cryptographic techniques** that has been used in history
  - **Classical cryptographic techniques** that are simple to understand and have been used at the early age of information systems.
  - **Basic concepts of modern cryptographic schemes**, including symmetric key cryptography and asymmetric key cryptography.
  - **Various techniques and mechanisms** to conduct symmetric key cryptography and asymmetric key cryptography.
  - **Difficulties** and **solutions** for secure key exchanges.
  - Application and beneficial cases of cryptography.
  - Techniques and Mechanisms to ensure **data integrity**.



Section 1:

Basic and Formal Definitions of Cryptography

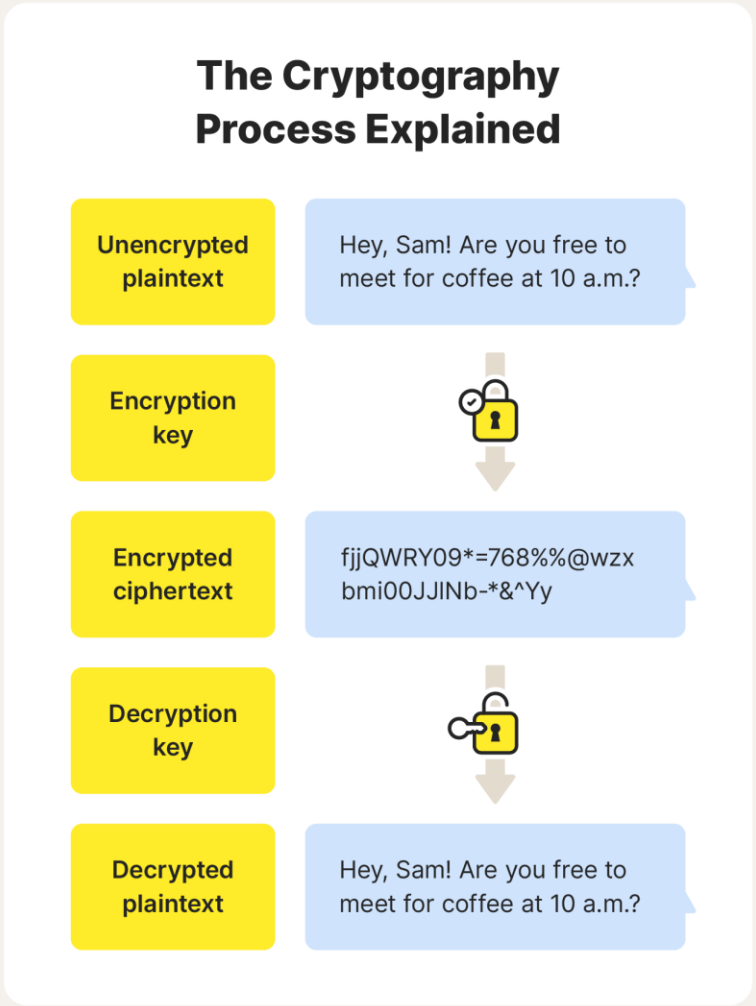
What is Cryptography?

- The mostly-used and well-known protection of information security (especially **Confidentiality**) is ***the application of cryptography***.

- **Cryptography** (*noun*):
  - It is derived from a Greek word, i.e., ***Kryptos***, meaning hidden.
  - The prefix '**crypto-**' means hidden or vault.
  - The suffix '**-graphy**' means writing.

Cryptography is **the process of hiding or coding information** so that only the person a message was intended for can read it

Basic Concept of Cryptography



- Cryptography involves two essential processes, including:

  - **Encryption** or **Encoding** or **Enciphering**
    - The process of encoding a message so that its meaning is not obvious.
    - We encrypt ***a plain text message*** (i.e., a readable text) to ***a cipher text*** (i.e., an unreadable or meaningless text).

  - **Decryption** or **Decoding** or **Diciphering**
    - The reverse process of encoding where we decode the cipher text back to the original plain text.

Formal Definition of Cryptographic Processes

- It is obvious that cryptographic processes are typically mathematical functions to transform input parameters to an output.
  - Suppose you have **a sentence (A)** that contains your data or information and you don't want to expose them to others.

    For example:

$$A = "I\ am\ 25\ years\ old"$$

  - We can consider the sentence (A) as **a plain text message (P)**, which could be represented by a sequence of characters:

$$P_A = \langle p_{A,1}, p_{A,2}, \dots, p_{A,n} \rangle = \langle I, a, m, 2, 5, y, e, a, r, s, o, l, d \rangle$$

  - To protect the plain text message against unauthorized access, it should be encrypted into **a cipher text (C)**:

$$C_A = \langle c_{A,1}, c_{A,2}, \dots, c_{A,m)} \rangle = \langle X, H, f, 2, d, e, R, w, 1, 3, 7, X, F, E, w, l, k, m, s, u, 1, j, d \rangle$$

  - $|P_A|$ and $|C_A|$ can be either in equal or different length ($n = m$ or $n \neq m$)

Formal Definition of Cryptographic Processes

- It is obvious that cryptographic processes are typically mathematical functions to transform input parameters to an output.
  - An **encryption process** is defined as a function $Encrypt(\cdot)$ or $E(\cdot)$ that takes a plain text message (P) and encode into a cipher text (C).

$$C = Encrypt(P) = E(P)$$

  - Reversely, a **decryption process** is defined as a function $Decrypt(\cdot)$ or $D(\cdot)$ that takes a cipher text (C) and decode it back to the original plain text message (P).

$$P = Decrypt(C) = D(C)$$

  - We can leverage **the transitive property** and rewrite the cryptographic processes as:

$$P = Decrypt\big(Encrypt(P)\big) = D(E(P))$$

  - **Drawback:** If you know the algorithm behind the functions, you will always be able to break it.

Formal Definition of Cryptographic Processes

- It is obvious that cryptographic processes are typically mathematical functions to transform input parameters to an output.
  - An **encryption process** is defined as a function $Encrypt(\cdot)$ or $E(\cdot)$ that takes a plain text message (P) with a key (K) and encode into a cipher text (C).

$$C = Encrypt(P, K) = E(P, K)$$

  - Reversely, a **decryption process** is defined as a function $Decrypt(\cdot)$ or $D(\cdot)$ that takes a cipher text (C) with a key (K) and decode it back to the original plain text message (P).

$$P = Decrypt(C, K) = D(C, K)$$

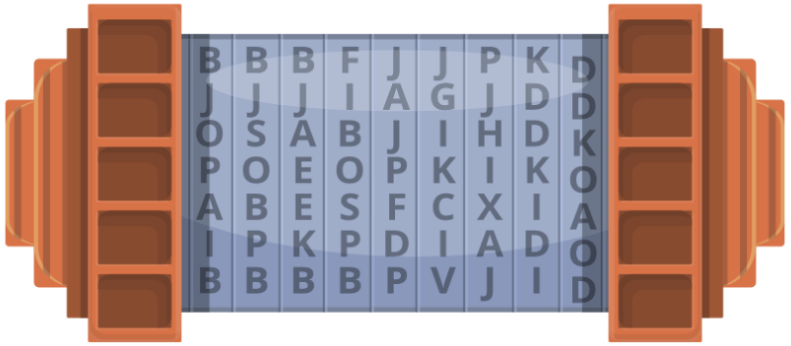
  - We can leverage **the transitive property** and rewrite the cryptographic processes as:

$$P = Decrypt(Encrypt(P, K), K) = D(E(P, K), K)$$

Operation and Performance of Cryptography

- Cryptographic processes are based on **two main operations**, where sometimes they are mixing:
  - **Substitution** – An operation to replace one item or character with another one (*Confusion*).
  - **Transposition** – An operation to change an order of items or characters (*Diffusion*).
- With these operations, it makes unauthorized entities harder to read the information. However, it can be exploited by brute-forcing the key space (Try every possible key).

- The ***performance/strength*** of the cryptographic processes depends on:
  - **The length of the key:** The longer the key length, the stronger the cryptography is. (*Take more time to guess!*)
  - **The work factor:** The more time and effort use, the stronger the cryptography is. (*One try take more time!*)

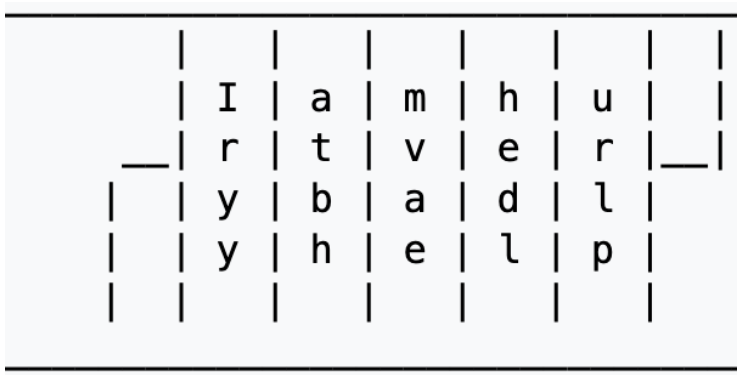


Section 2:

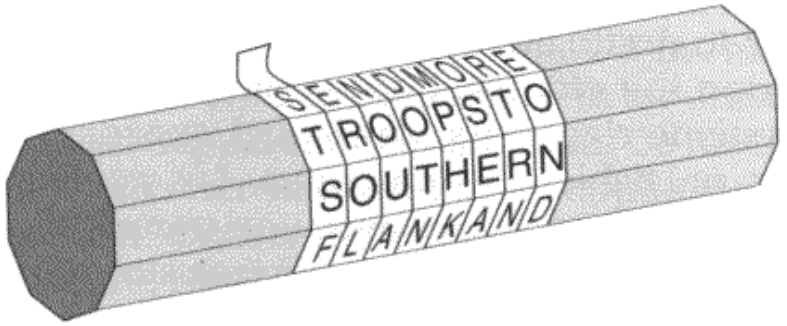
Early Cryptographic Techniques in History

Cryptographic Techniques in History

- **Scytale Cipher or Spartan Cipher**
  - It is a tool to perform a transposition cipher back in the age of Greek and Spartan, consisting of:
    - A ***cylinder*** that is agreed upon both communication parties on the diameter.
    - A ***wound parchment*** where a message can be written on.
  - The recipient uses a rod or a cylinder of the same diameter on which the parchment is wrapped to read the message.

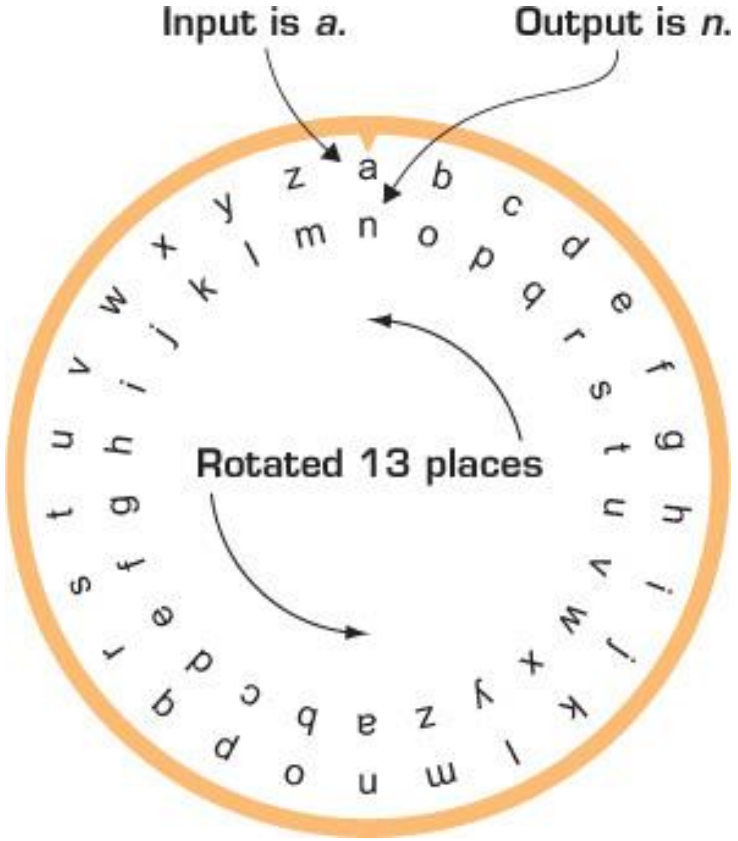


Cipher Text
"Iryyatbhmvaehedlurlp"



Cryptographic Techniques in History

- **Shift Cipher or Roman's Substitution Cipher**
  - It is a tool to perform an encryption by shifting alphabets by a certain number of locations or offsets.
  - **Examples of Shift Cipher:**
    - **Rotate13 or ROT13 Cipher**, i.e., a cipher that shifts the position by 13, such as $a \rightarrow n$ and $h \rightarrow u$.
    - **Caesar Cipher**, i.e., a cipher that sets a key as an integer number and shifts the position of the inner wheel by the key, such as the key = 10 then $a \rightarrow j$.



Cryptographic Techniques in History

- **Other Ciphers in Real Life**

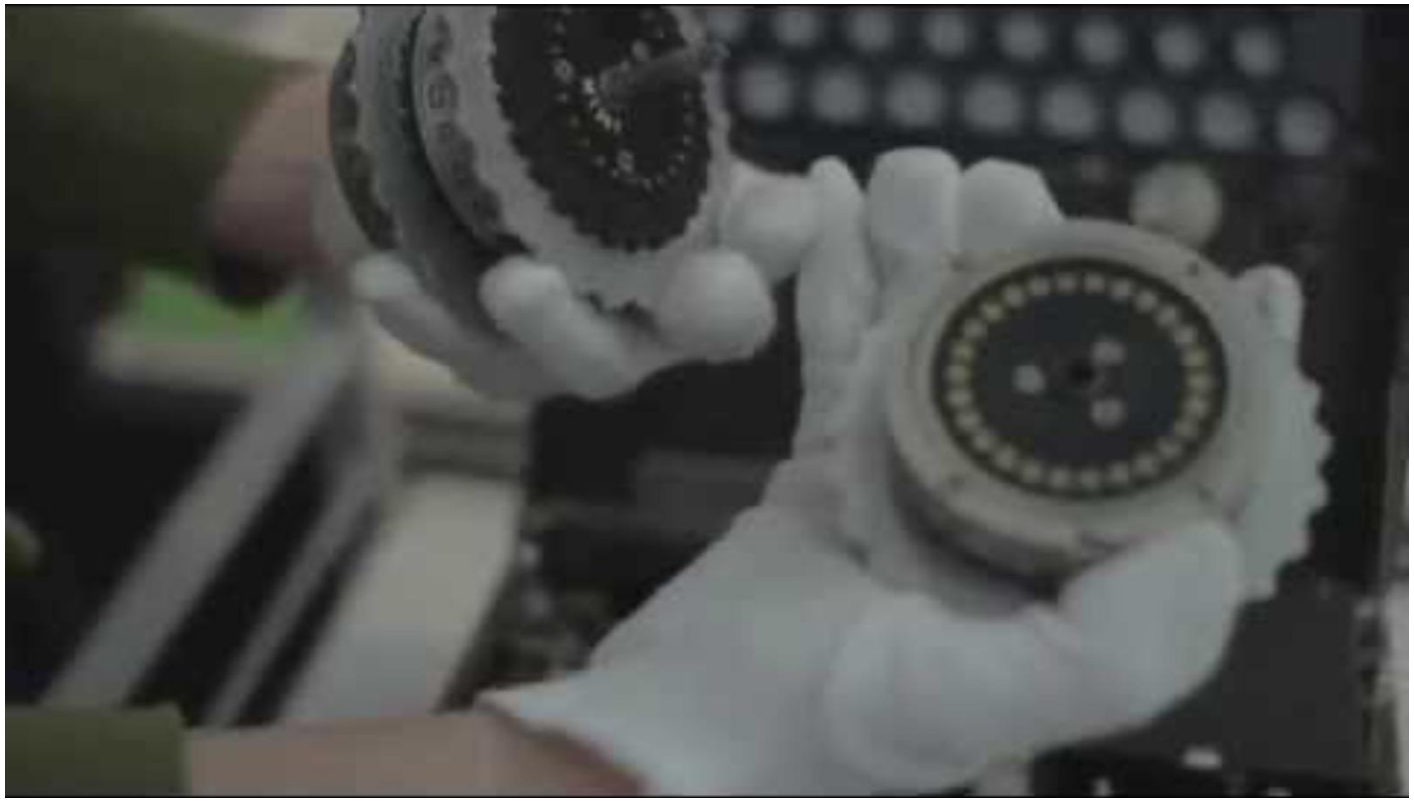
Morse Code

Engima Machine

Slang/Idioms

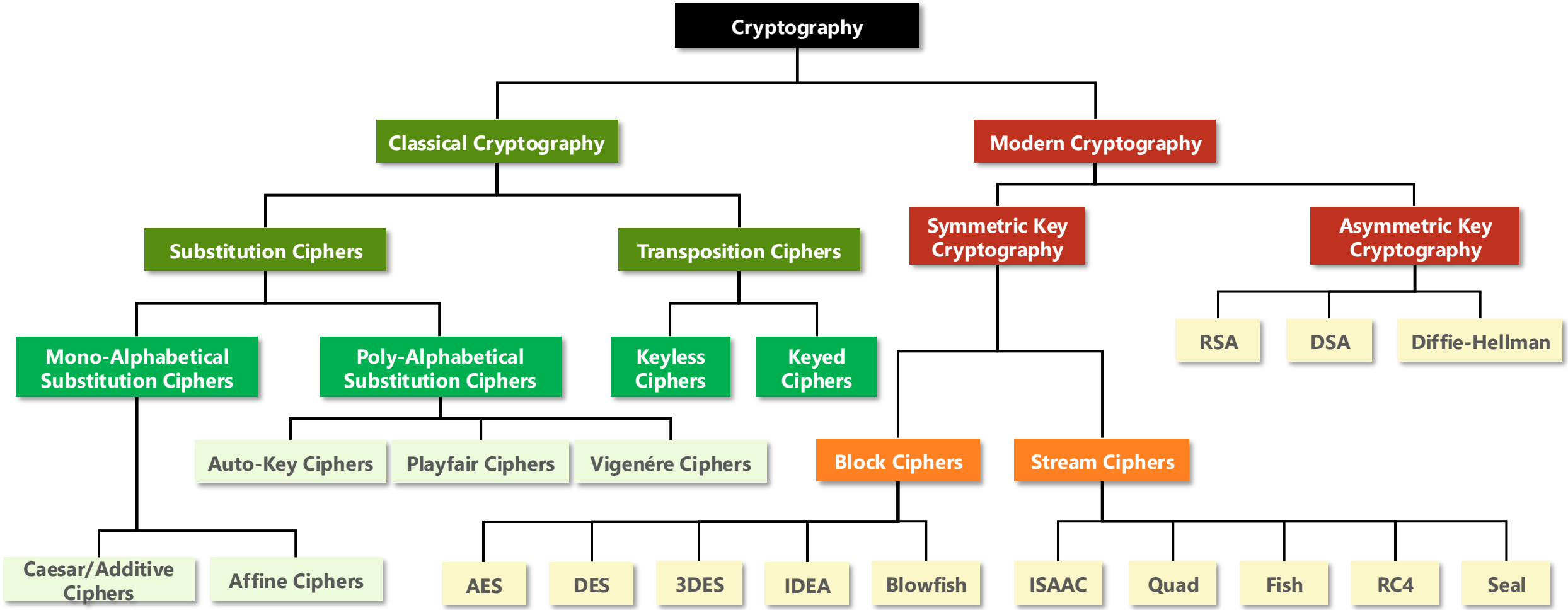
Cryptographic Techniques in History

- **Other Ciphers in Real Life**



Source: https://www.youtube.com/watch?v=TYX691q2J2c

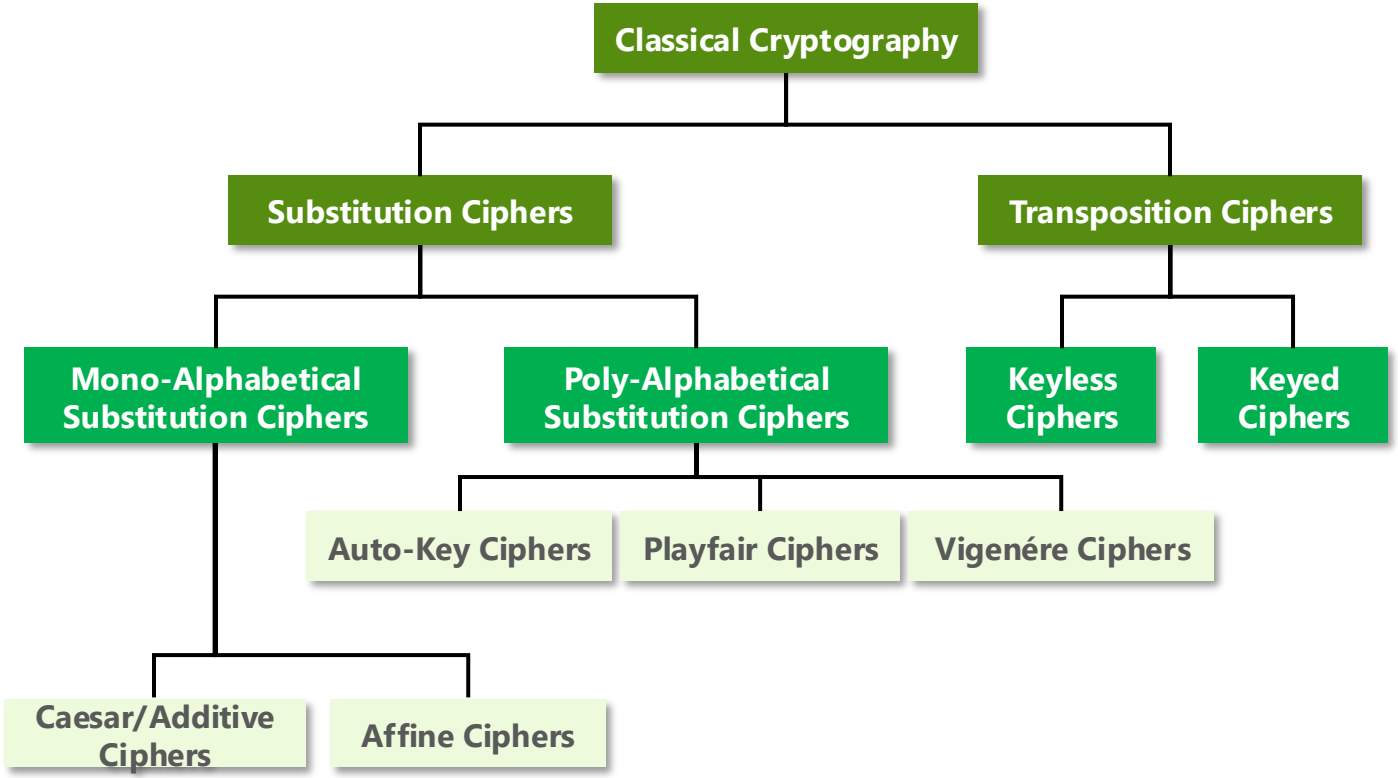
Classification of Cryptographic Techniques



Section 3:

Classical Cryptographic Techniques

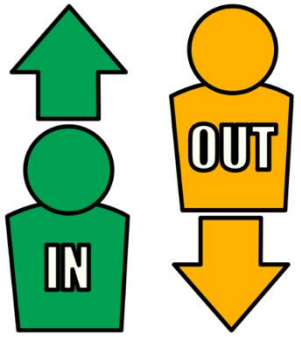
Classical Cryptographic Techniques



Substitution Ciphers

- Substitution ciphers are a group of cryptographic techniques or schemes that aim to substitute every character in a plain text message ($\forall p \in P$) wuth a different cipher character.

- There are ***two main groups*** of substitution ciphers:
  - **Mono-alphabetical substitution ciphers** → *Additive Cipher* and *Affine Cipher*
    - Every occurrence of a character will be substituted by a specific character (1-to-1 substitution)
  - **Poly-alphabetical substitution ciphers** → *Auto-key Cipher*, *Playfair Cipher*, and *Vigenère Cipher*
    - Each occurrence of a character will be substituted by different characters.



Substitution Ciphers: Mono-Alphabetical Ciphers

Additive Cipher

1. Each character in the plain text message will be encoded in a *decimal* format.
2. Then, the encoded value of each character will be **added by a key value** from a range of 0 – 25.
3. The addition result will be *modulo-ed by 26*.
4. The addition result will be used to represent another character, which is considered as a cipher character.

Example:

Plain Text Message: H E L L O
(H=7, E=4, L=11, O=14)

Key Value: 15

Cipher Text: (7+15, 4+15, 11+15, 11+15, 14+15)
(22%26, 19%26, 26%26, 26%26, 29%26)
(22, 19, 0, 0, 3) = W T A A D

Affine Cipher

1. Each character in the plain text message will be encoded in a *decimal* format.
2. Then, the encoded value of each character will be **calculated with some mathematical functions against the key value**.
3. The calculated result will be *modulo-ed by 26*.
4. The calculated result will be used to represent another character, which is considered as a cipher character.

Example:

$$C = (P + K)\%26$$
$$P = (C - K)\%26$$

$$C = (K_1 P + K_2)\%26$$
$$P = \left(\frac{C - K_2}{K_1}\right)\%26$$

Substitution Ciphers: Mono-Alphabetical Ciphers

Plaintext:
upon this basis i am going to show you how a bunxh of bright young folks did find a xhampion a man with boys and girls of his own.
Key:
G A B S L Y T E X U C F H I J K Z M N O P Q R D V W
Cryptogram:
pkji oexn agnxn x gh tjxit oj nejr vjp ejr g apide jy amx teo vjpit yjfcn sxs yxis g deghkxji g hgi rxoe ajvn gis txmfn jy exn jri.

Key

Plaintext:	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ciphertext:	CRYPTOGRAM56789BDEFHIJKLNQ

1	2	3	4	S
U	V	W	X	
	Z			

Message

Plaintext:	THIS IS A SECRET MESSAGE
Ciphertext:	HRAF AF C FTYE2H 7VFF1GZ

Weakness: Frequency analysis can be used to defeat this kind of ciphers/cryptographic techniques. How?

Substitution Ciphers: Poly-Alphabetical Ciphers

Auto-Key Cipher

1. Each character in the plain text message will be encoded in a *decimal* format.
2. A key will be set as a decimal number (any number).
3. The encoded value of each character will be added with **the key stream**.
4. A key is used to generate a key stream by:
   - The **first** element of the key stream will be identical to **the pre-defined key value**.
   - The **second** element of the key stream will be taken **from the encoded value of the first character** in the plain text message.
   - The **third** element of the key stream will be taken from **the encoded value of the second character** in the plain text message. And so on...

Example:

Plain Text Message:	A	T	T	A	C	K	
	0	19	19	0	2	10	
Key Value:	12						+
Key Stream:	12	0	19	19	0	2	
Cipher Text:	12	19	12	19	2	12	
	M	T	M	T	C	M	

Plain Text Message:	R	A	I	N	I	N	G
	17	0	8	13	8	13	6
Key Value:	7						
Key Stream:	7	17	0	8	13	8	13
Cipher Text:	24	17	8	21	21	21	19
	Y	R	I	V	V	V	T

Substitution Ciphers: Poly-Alphabetical Ciphers

Playfair Cipher

1. Each character in the plain text message will be encoded in a *decimal* format.
2. An encryption key is generated from 26 unique English alphabets arrange in a 5×5 **matrix**.
3. **Two characters in the plain text message will be paired up** under conditions that (1) two characters in a pair must not be the same, and (2) a random character can be padded in the end of the message if the plain text message is in an odd length.
4. The encryption will be processed for each pair based on the following rules:
   - **Rule 1:** If two characters in the pair are in the same row of the matrix, then c will be the one next to the right of each.
   - **Rule 2:** If two characters in the pair are in the same column of the matrix, then c will be the one below of each.
   - **Rule 3:** If two characters in the pair are neither in the same row nor column, then c will be the one in the same row of the character but in the same column of another character in the pair.

Example:

Key Matrix $=$

$$\begin{bmatrix} L & G & D & B & A \\ Q & M & H & E & C \\ U & R & N & I/J & F \\ X & V & S & O & L \\ Z & Y & W & T & P \end{bmatrix}$$

	(H, E)	(L, X)	(L, O)
Plain Text Message: HELLO	H → E E → C	L → Q X → Z	L → B O → X
	Rule 1	**Rule 2**	**Rule 3**

Cipher Text: EXQZVX

Substitution Ciphers: Poly-Alphabetical Ciphers

Vigenère Cipher

1. Each character in the plain text message will be encoded in a *decimal* format.
2. **The key stream** is generated from **the repetition of the initial secret key (K)** of length m $(1 \leq m \leq 26)$
3. The encryption is done by a simple mathematical function between the encoded plain text message and the key stream

Example:

Plain Text Message:	A	B	C	D	E	F	
	0	1	2	3	4	5	
Key Value: 5, 8, 0 — **Key Stream:**	5	8	0	5	8	0	+
Cipher Text:	5	9	2	8	12	5	
	F	J	C	I	M	F	

Transposition Ciphers

- The transposition cipher encrypts a plain text message by shifting characters to a specific offset.
- There are **two main groups** of transposition ciphers, which are:

Keyless Transposition Cipher

- There are two simple methods to transpose the plain text message to be encrypted.
  1. Write column by column, but send row by row.
  2. Write row by row, but send column by column.

Plain Text Message (P) = "meet me at the park"

m	t	a	h	a
e	m	t	e	r
e	e	t	p	k

C = "mtahaemtereetpk"

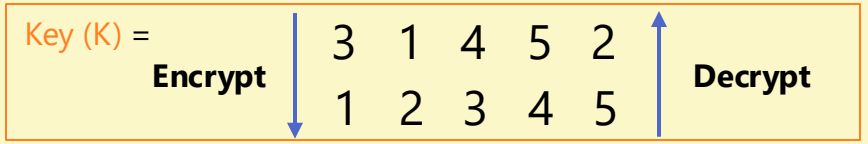
m	e	e	t	m
e	a	t	t	h
e	p	a	r	k

C = "meeeapetattrmhk"

Keyed Transposition Cipher

- A plain text message is decomposed into small chunks of characters and permuted per chunk.

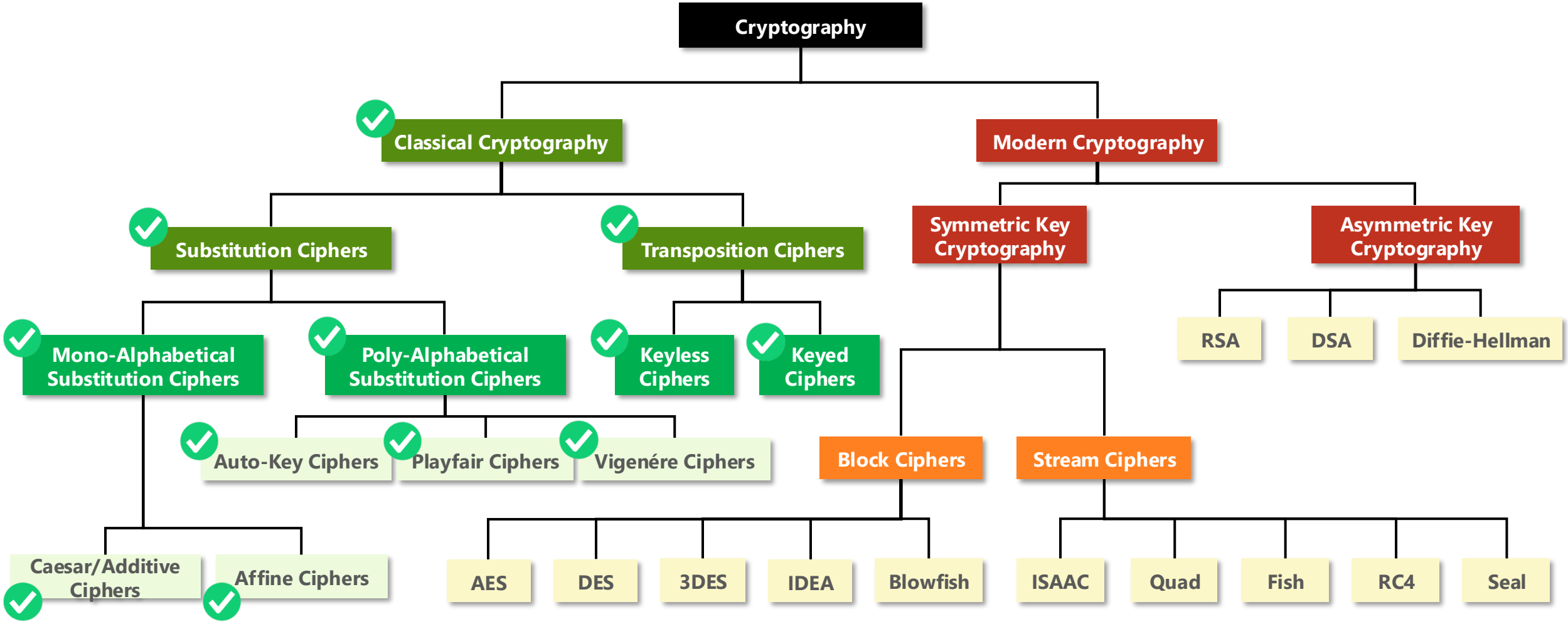
Plain Text Message (P) = "enemy attack at night"

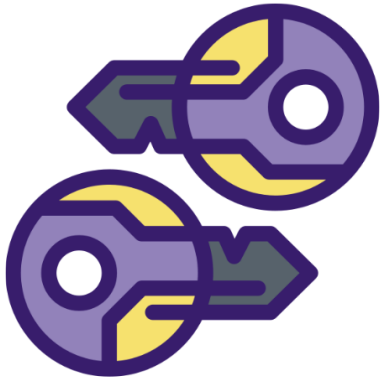


E N E M Y | A T T A C | K A T N I | G H T X Y

E E M Y N | T A A C T | T K N I A | T G X Y H

Classification of Cryptographic Techniques

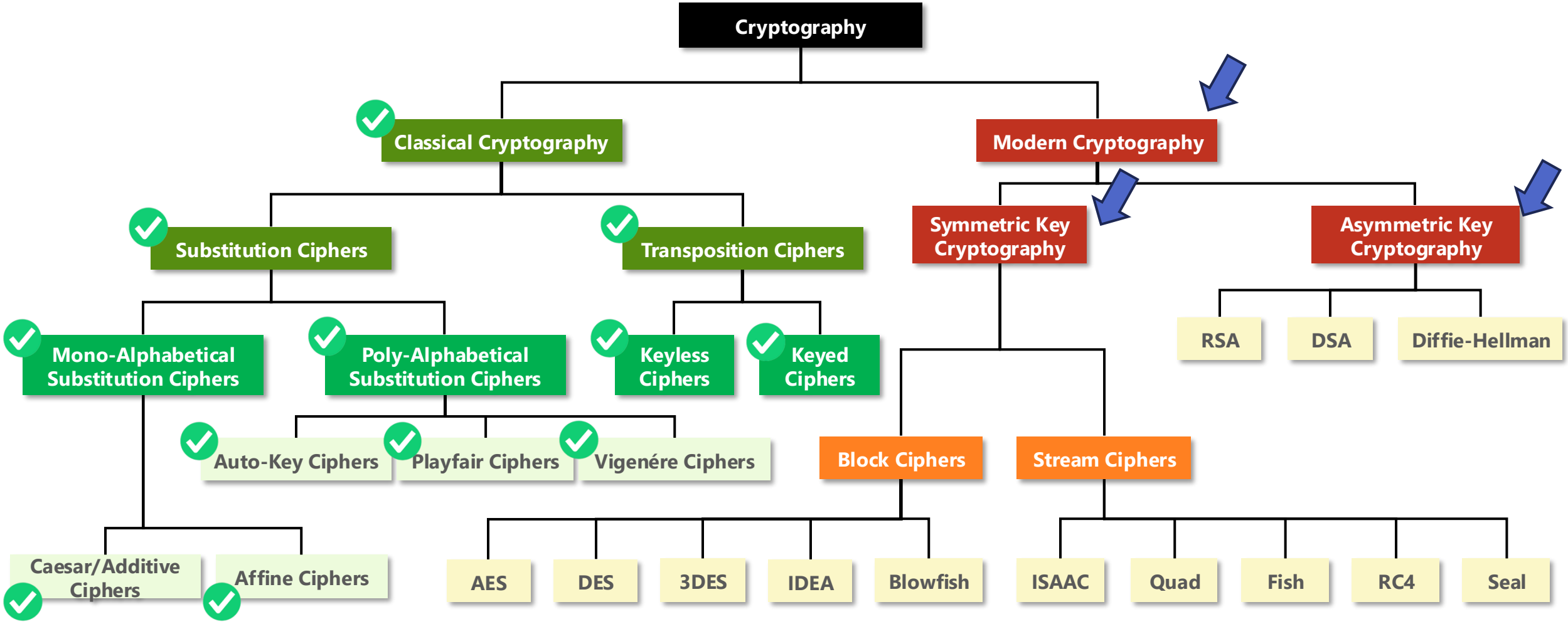




Section 4:

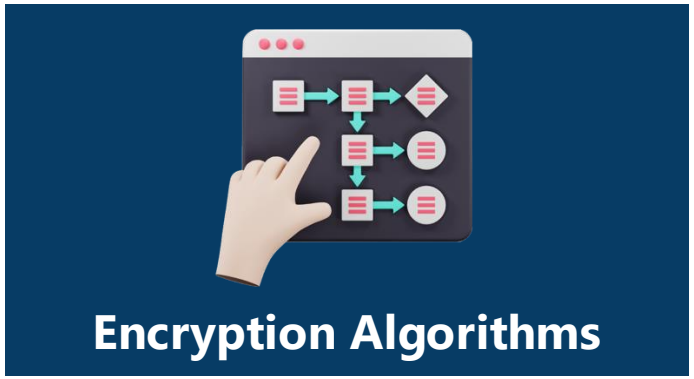
Modern Cryptographic Techniques

Classification of Cryptographic Techniques



An Introduction to Modern Cryptographic Techniques

- We have learned from the classical cryptographic techniques that we can introduce confusion and diffusion to the plain text message in order to make it unintelligible.
- However, we might realize that the techniques we discussed so far are simple and easy to break.
- There are **three common components** in cryptographic techniques:



- **Modern cryptography** is an **enhancement** of ***encryption algorithms*** to the stage that it is trustworthy to be implemented in modern information systems.

An Introduction to Modern Cryptographic Techniques

- An ***encryption algorithm*** is a systematic **method** used to transform **data** (not just text!) into ciphertext in **a predictable way** by using **an encryption key**. Also, ciphertext can be transformed back to the original data by using **a decryption key**.

Question: Do you think substitution ciphers and transposition ciphers we have learned in the previous lecture are encryption algorithms?

Question: Are substitution and transposition ciphers "**Good**"?

Characteristics of Good Ciphers

- Back in 1949, Claude Shannon proposed characteristics for identifying a good cipher as:
  1. The ***amount of secrecy needed should determine the amount of labor*** appropriate for the encryption and decryption.
  2. The set of ***keys*** and the enciphering ***algorithm*** should be ***free from complexity***.
  3. The **implementation** of the process should be ***as simple as possible***.
  4. ***Errors*** in ciphering **should not propagate** and cause corruption of further information.
  5. The ***size*** of the enciphered text ***should be no larger*** than the text of the original message.

Characteristics of Trustworthy Ciphers

- Apart from the characteristics of good ciphers, encryption algorithms should also be trustworthy and "commercial grade," meaning that they should meet the following constraints:
    1. The encryption algorithm is based on **sound mathematics**.
    2. The encryption algorithm has been **analyzed thoroughly by competent experts** and found to be sound and trustworthy.
    3. The encryption algorithm has stood the "**test of time**."
        - As time passes, the encryption algorithm should still be usable.

Classification of Modern Cryptography

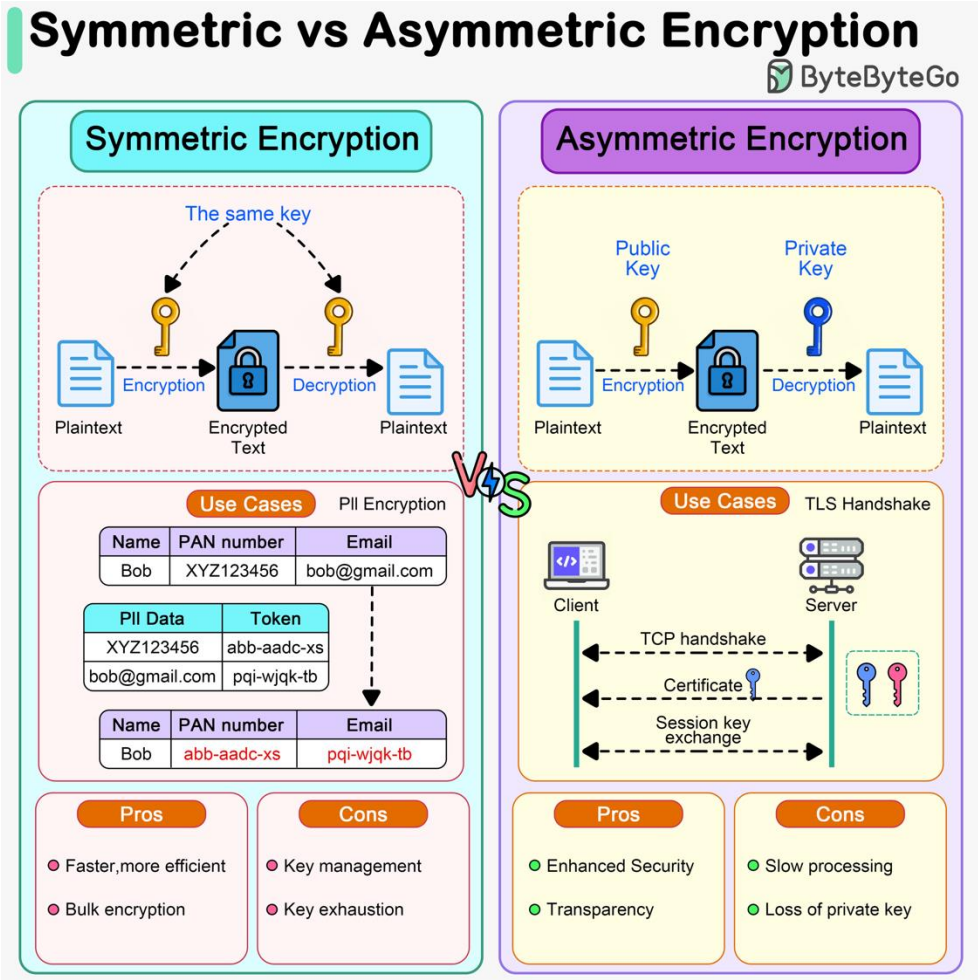
- In the modern cryptography paradigm, there are **two groups** of encryption algorithms:

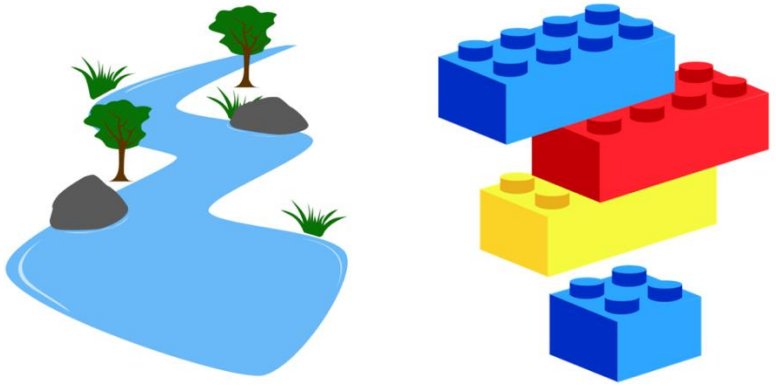
1. **Symmetric Key Cryptography:** Both parties (i.e., encryptor and decryptor) use the same encryption key in this group of algorithms.

   Sub-types: Stream Ciphers, Block Ciphers

2. **Asymmetric Key Cryptography:** The encryptor uses a unique key that is different from the one used by the decryptor.

   Sub-Types: RSA, DSA, Diffie-Hellman





Section 4.1:

Symmetric Key Cryptography

Basic Concept of Symmetric Key Cryptography

- **Symmetric Key Cryptography:**
  - A.K.A. *Private-Key* or *Secret-Key* Encryption.
  - The ***same key*** is used for encryption and decryption.
  - Applying an algorithm with the private key to make them ***unintelligible***.
  - The simplest algorithm (such as ***an exclusive OR*** operation) can make the encryption nearly tamper-proof.

Encryption: $(Key, Plain\ Text) \rightarrow Cipher\ Text$

Decryption: $(Key, Cipher\ Text) \rightarrow Plain\ Text$

Correctness Property:

$$Decrypt\big(K, Encrypt(K, P)\big) = P$$

A Theory of Secure Symmetric Key Cryptography

Shannon's Theorem

A private key cryptosystem will be **completely secure** if the key is at least as long as the message to be encrypted.

- It requires a secure channel to exchange the key between parties (**Problem?**).

- The **main shortcoming** of symmetric key cryptography is how keys are exchanged.
  - A key must be sent in **plaintext form** since there is no key for encryption of the key.
  - Users have to use as many private keys as there are people to communicate with.

    For instance, to communicate with N people, it is required to distribute keys equal to $\frac{N*(N-1)}{2}$ times.

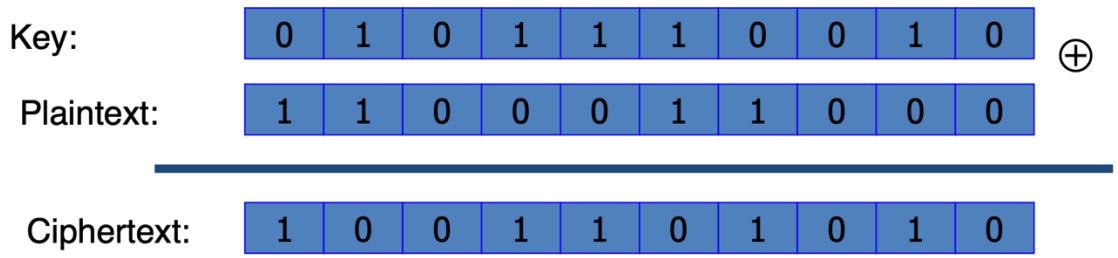
Section 4.1.1:

Stream Ciphers

One-Time Padding (OTP) Encryption Scheme

- The original idea of stream ciphers came from an old encryption scheme called **the One-Time Pad (OTP)**, developed by Vernam in 1917.
- In the OTP scheme, encryption starts with **a key (K)**, which is *a uniformly random sequence of bits having the same length* as the plaintext message.
- To encrypt the message, each byte (8 bits) of the plaintext is encrypted by **applying the XOR operation** with the corresponding byte from the key.

Vernam (1917)



- **Encryption:** $C = E(K, P) = K \oplus P$
- **Decryption:** $P = D(K, C) = K \oplus C$
  $= K \oplus (K \oplus P) = P$

p	q	$p \oplus q$
0	0	0
0	1	1
1	0	1
1	1	0

One-Time Padding (OTP) Encryption Scheme

- The original idea of stream ciphers came from an old encryption scheme called **the One-Time Pad (OTP)**, developed by Vernam in 1917.
- In the OTP scheme, encryption starts with **a key (K)**, which is *a uniformly random sequence of bits having the same length* as the plaintext message.
- To encrypt the message, each byte (8 bits) of the plaintext is encrypted by **applying the XOR operation** with the corresponding byte from the key.

Encoded Plain Text (P): 10011001
Key (K): 11000011 ⊕

Cipher Text (C): 01011010

Cipher Text (C): 01011010
Keys (K): 11000011 ⊕

Original Plain Text (P): 10011001

p	q	$p \oplus q$
0	0	0
0	1	1
1	0	1
1	1	0

One-Time Padding (OTP) Encryption Scheme

- **Example:** I would like to encrypt a plaintext message "**OK**" with the stream cipher with **a key 1101 0011**.

1. I encoded each character in the plaintext message in decimal format. "O" = 14 and "K" = 10
2. I converted each decimal number into an 8-bit binary format. "O" = 0000 1110 and "K" = 0000 1010
3. I performed the XOR operation using the key.

$$\begin{array}{lll} \text{"O"} = 0000\ 1110 & \text{and} & \text{"K"} = 0000\ 1010 \\ K = 1101\ 0011 \oplus & & K = 1101\ 0011 \oplus \\ 1101\ 1101 & & 1101\ 1001 \end{array}$$

4. I converted the resulting binary values back to decimal numbers.

1101 1101 = 219 % 26 = 11 = "L"

1101 1001 = 217 % 26 = 9 = "J"

5. As a result, the ciphertext of the plaintext message 'OK' is '**LJ**'.

A	B	C	D	E	F	G	H	I	J	K	L	M
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
13	14	15	16	17	18	19	20	21	22	23	24	25

p	q	$p \oplus q$
0	0	0
0	1	1
1	0	1
1	1	0

Issues with the OTP Encryption Scheme

- **What do you think they are issues of the OTP cryptographic scheme?**
- **Problem 1:** A key must be generated with the same length as the plaintext message every time you want to encrypt it. How can we know the length of the plaintext message beforehand?

- **Why do we call this scheme as One-Time?**
- **Problem 2:** A key can be used only once!

$$C_1 = P_1 \oplus K$$

$$C_2 = P_2 \oplus K$$

$$C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K) = P_1 \oplus P_2$$

Let's Try!

P_1 = 51 = 00110011
P_2 = 17 = 00010001
K = 244 = 11110100

00110011 ⊕ 11110100 = 11000111

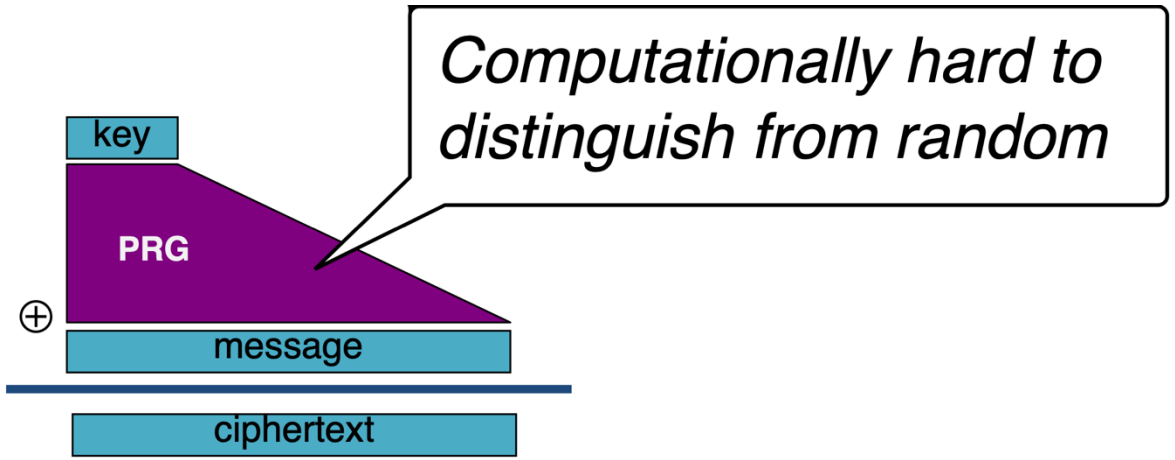
00010001 ⊕ 11110100 = 11100101

11000111 ⊕ 11100101 = **00100010**

00110011 ⊕ 00010001 = **00100010**

Stream Ciphers

- **Stream ciphers** are an enhancement of the OTP encryption scheme, designed to *expand the key length* to match the length of the plaintext.
- **Stream ciphers** encrypt the plaintext **byte by byte** *(8 bits = 1 byte)*.
- **Stream ciphers** address the problem of not knowing the exact length of the plaintext in advance.
- **Stream ciphers** use a short key to generate **a longer keystream** using a pseudorandom generator (PRG).



OTP: $C = P \oplus K$ $P = C \oplus K$

Stream Cipher: $C = P \oplus PRG(K)$

$P = C \oplus PRG(K)$

Issues with the Stream Ciphers

- **What do you think they are issues of the OTP cryptographic scheme?**
- **Problem 1:** A key must be generated with the same length as the plaintext message every time you want to encrypt it.
  How can we know the length of the plaintext message beforehand? ✅ **Solved!**

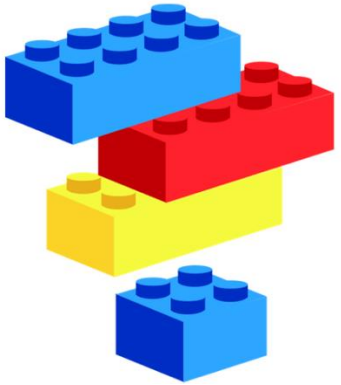
- **Why do we call this scheme as One-Time?**
- **Problem 2:** A key can be used only once!

$$C_1 = P_1 \oplus PRG(K)$$

$$C_2 = P_2 \oplus PRG(K)$$

➡

$$C_1 \oplus C_2 = \big(P_1 \oplus PRG(K)\big) \oplus (P_2 \oplus PRG(K)) = P_1 \oplus P_2$$

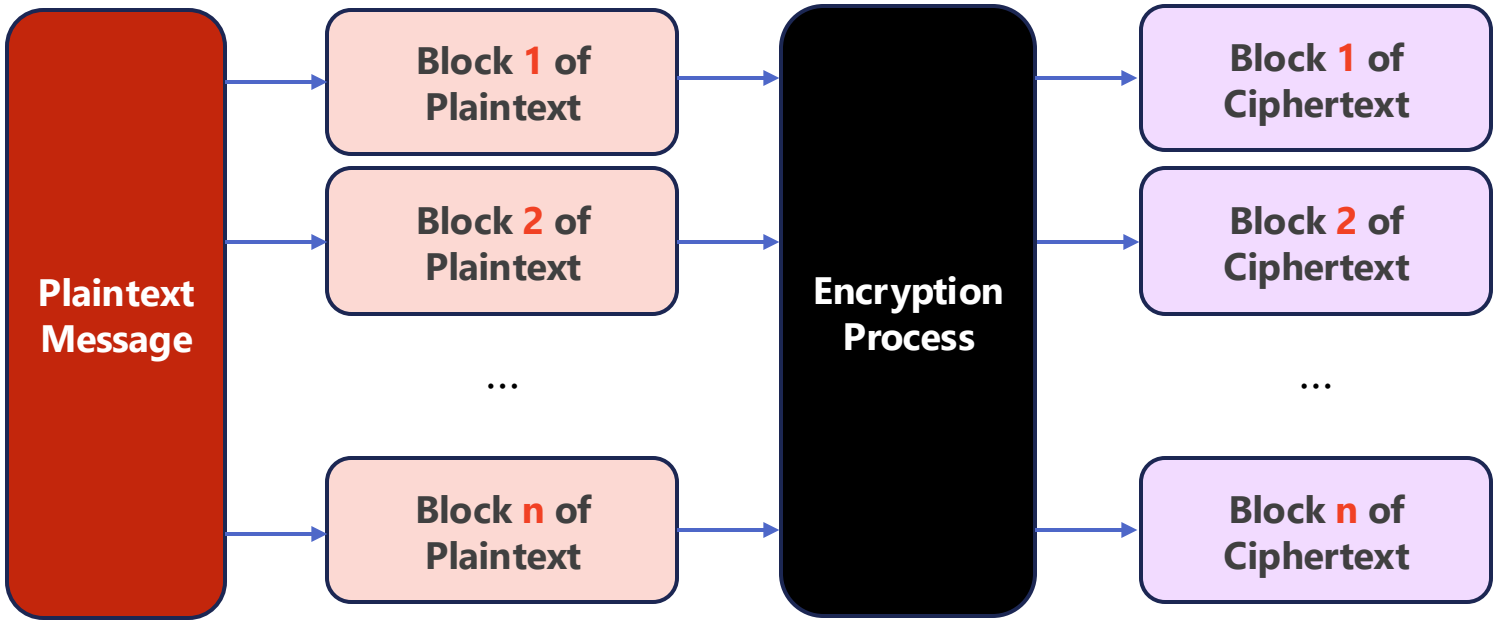


Section 4.1.2:

Block Ciphers

An Introduction to Block Ciphers

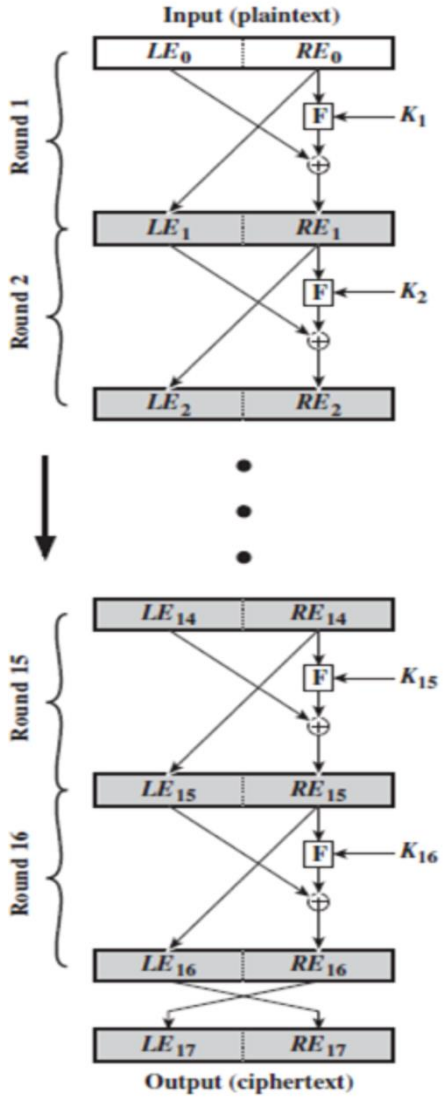
- **Block ciphers** are another type of symmetric key cryptography that divide the input plaintext into fixed-size blocks, with the encryption algorithm applied to each block individually.
- Different block cipher schemes may use **different block sizes** (e.g., 64-bit, 128-bit).
- The encryption key size must be compatible with the block size.



Variations of Block Ciphers

- Data Encryption Standard (DES)
- Triple DES (3DES)
- Advanced Encryption Standard (AES)
- Ron's Code/Rivest Cipher (RC2, RC6)
- Blowfish

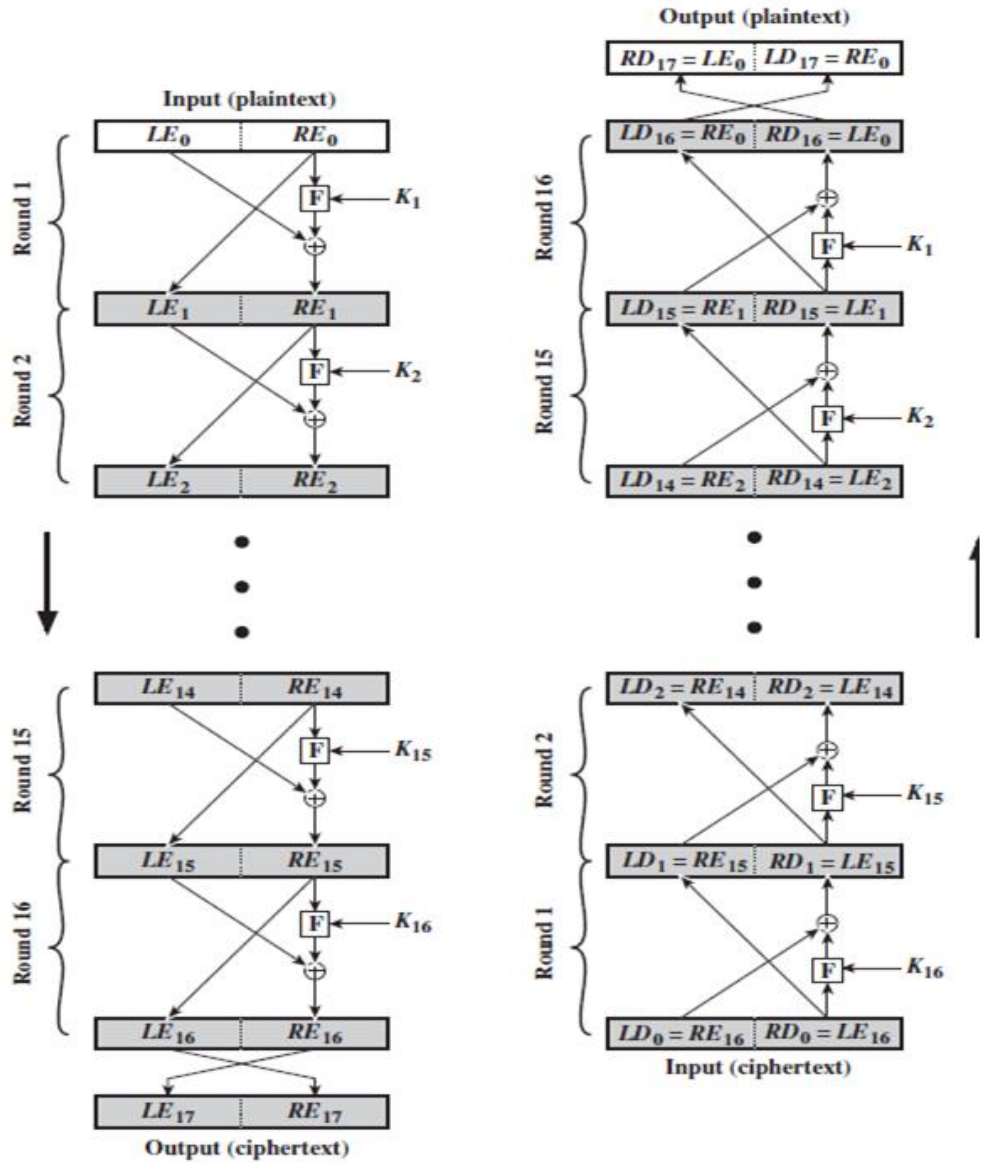
Feistel (F) Network

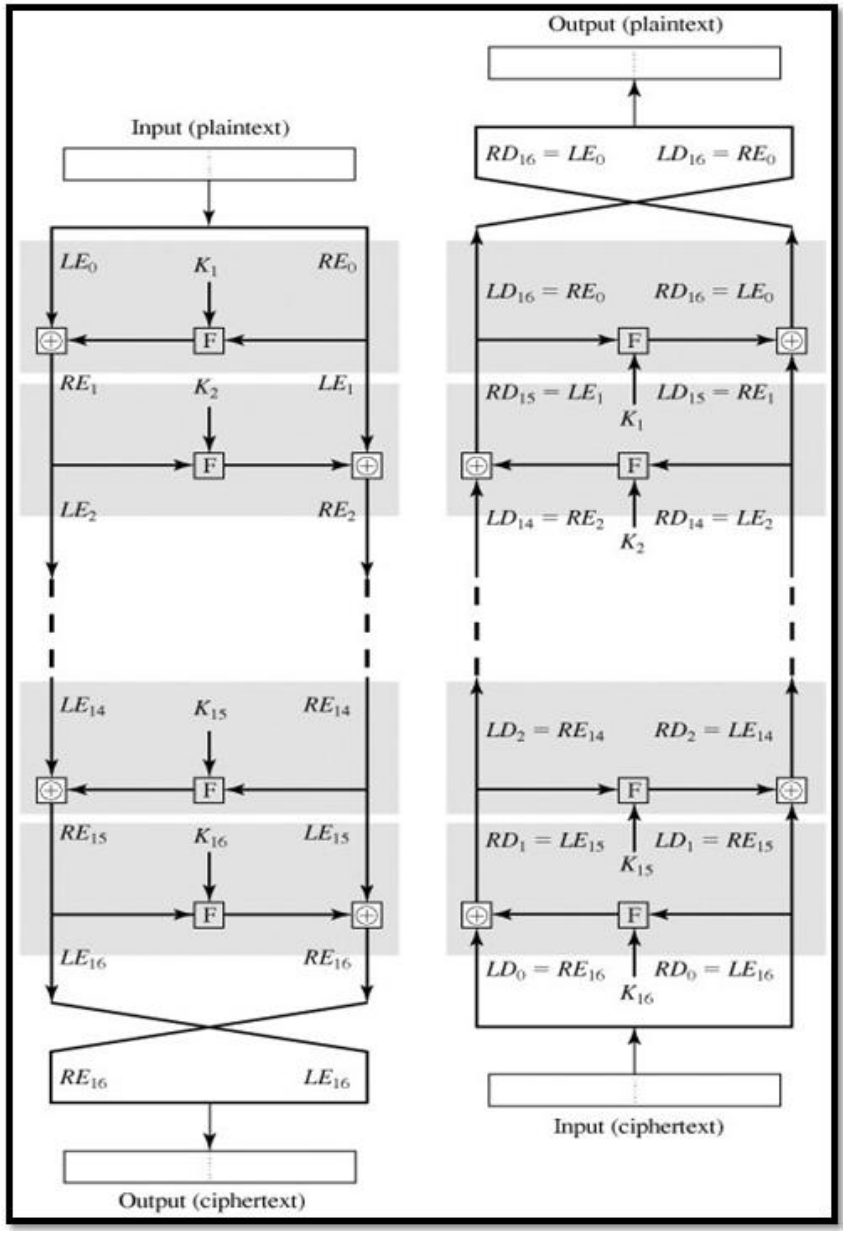


To understand block ciphers, let's start with the original cipher.

- **Feistel (F) Network or Feistel Cipher** is Feistel's proposal to produce a block cipher using the concepts of ***permutation*** and ***substitution***.
- The inputs to the encryption algorithm are:
  - **A plaintext block of length 2w bits** (i.e., 2-word bits), which can be divided into two halves:
    - LE_0 is the left half of the plain text block in the initial round,
    - RE_0 is the right half of the plain text block in the initial round.
  - **A key block (K_i) for the *i*th round.** The key block is changed in each round.

Feistel (F) Network





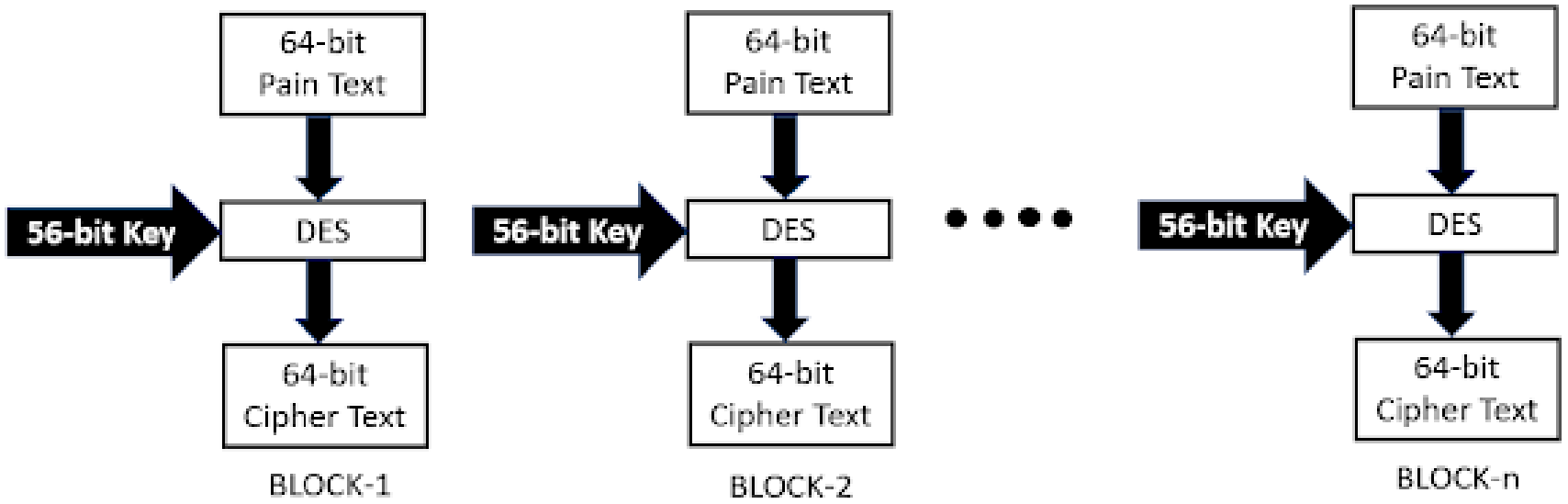
Data Encryption Standard (DES)

- **Data Encryption Standard (DES)** was developed at IBM in 1972 and based on an early design by *Horst Feistel*. The algorithm was submitted to the **National Bureau of Standards (NBS)** following the agency's invitation to propose a candidate for the protection of sensitive, unclassified electronic government data.

- In 1976, after consultation with the National Security Agency (NSA), the NBS selected a slightly modified version of DES (i.e., strengthened against differential cryptanalysis, but weakened against brute-force attacks), which was published as an official Federal Information Processing Standard (FIPS) for the United States in 1977.

Data Encryption Standard (DES)

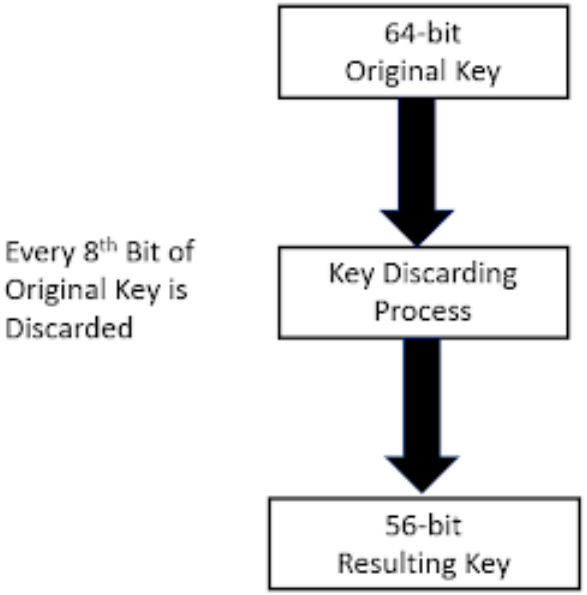
- **Data Encryption Standard (DES)** has been recognized as an official cryptographic standard both in the U.S. and abroad since 1977. DES is a symmetric cipher algorithm and uses the block cipher method for encryption and decryption.



- Normally, DES requires **two inputs: 64-bit plain text message blocks**, and **a 64-bit key**.

Data Encryption Standard (DES)

- The **original key** of DES is **64 bits**, but the algorithm requires just **a 56-bit key**:
  - Prior to the initialization of encryption algorithm, we ***must discard 8 bits of original key***.

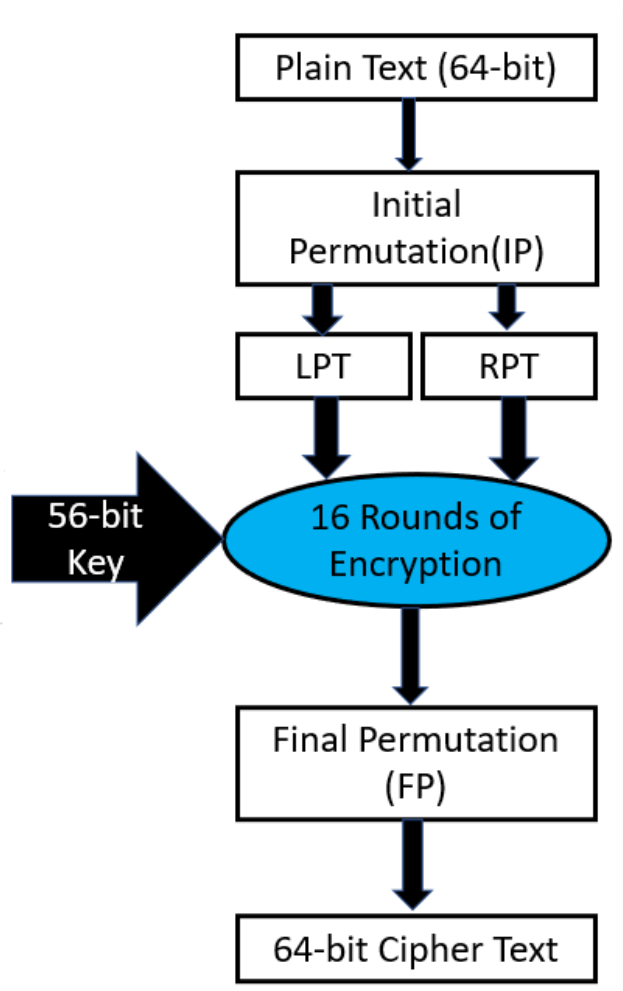


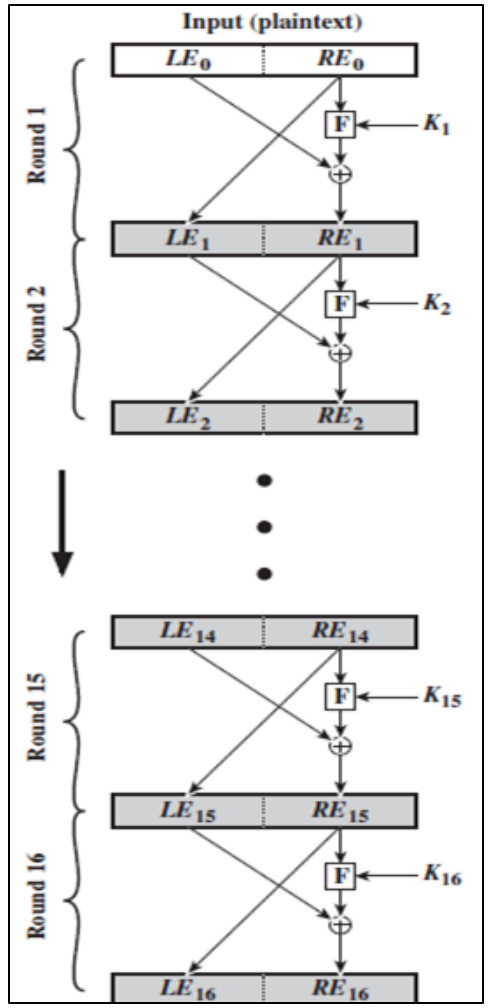
1	2	21	38	58	15	37	26
22	55	44	3	53	27	11	60
49	28	14	42	61	48	63	41
18	39	56	10	64	16	62	8
45	40	20	54	4	33	34	52
7	30	47	59	32	5	35	25
29	12	13	6	24	46	57	36
17	23	50	31	43	51	9	19

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

Data Encryption Standard (DES)

- **DES algorithm** consists of **five steps**:
  - **Step 1:** 64-bit input plain text message is given to the **Initial Permutation (IP)** function.
  - **Step 2:** IP function produces two halves of the permuted block known as Left Plain Text (LPT) and Right Plain Text (RPT).
  - **Step 3:** Each pair of LPT and RPT performs **16 rounds of encryption process**.
  - **Step 4:** Resulting LPT and RPT are rejoined, and the **Final Permutation (FP)** is performed on the combined block.
  - **Step 5:** 64-bit cipher text block is generated.

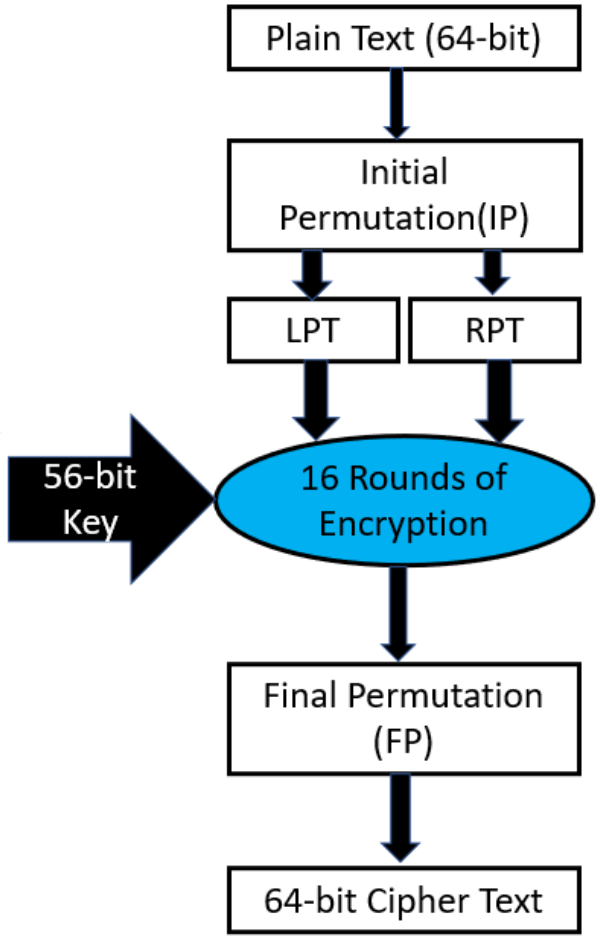




Data Encryption Standard (DES)

- **Step 1**: 64-bit input plain text message is given to the **Initial Permutation (IP)** function.
  - The IP function is performed only once.
  - Bit sequence of the input plain text message block is reordered according to the predefined IP table.
  - For example, the 1st bit takes a value from the 40th bit, or the 58th bit take a value from the 1st bit.

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

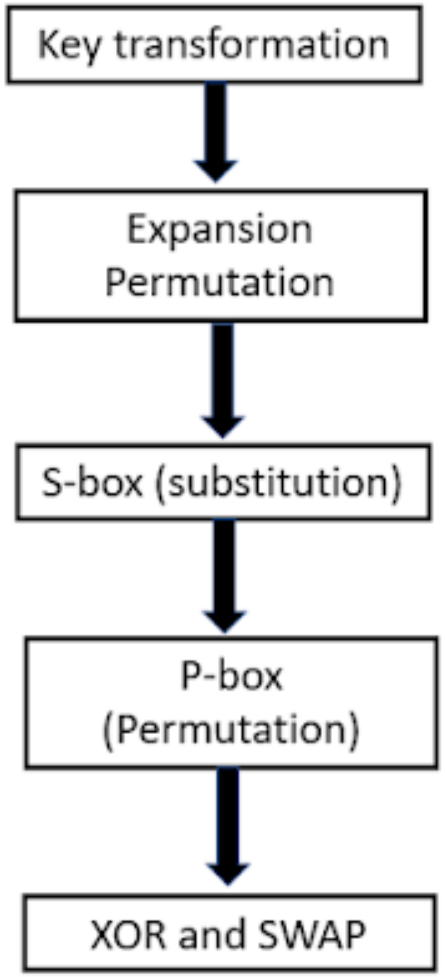


- **Step 2:** IP function produces two halves of the permuted block known as Left Plain Text (LPT) and Right Plain Text (RPT).

Data Encryption Standard (DES)

- **Step 3:** Each pair of LPT and RPT performs **16 rounds of encryption process**.
  - **Step 3.1: Key Transformation (56-bit Key)**
    - Key bits shifted per round
    - Compression Permutation
  - **Step 3.2: Expansion Permutation of Plain Text and XOR**

    (Plain Text Size: 48 bits, Cipher Text Size: 48 bits)
  - **Step 3.3: S-Box Substitution**
  - **Step 3.4: P-Box Permutation**
  - **Step 3.5: XOR and Swap**



Data Encryption Standard (DES)

- **Step 3:** Each pair of LPT and RPT performs **16 rounds of encryption process**.
  - **Step 3.1: Key Transformation (56-bit Key): Key bits shifted per round**
    - The 56-bit key is divided into **two halves, each of 28 bits**.
    - **Circular shifts** are applied to **each half**.
    - The number of bit shifts depends on the encryption round:
      - In the 1st, 2nd, 9th, and 16th rounds, each half is *shifted left* by **one position**.
      - In all other rounds, each half is *shifted left* by **two positions**.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Key bit shifted	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Data Encryption Standard (DES)

- **Step 3:** Each pair of LPT and RPT performs **16 rounds of encryption process**.
  - **Step 3.1: Key Transformation (56-bit Key): Compression Permutation**
    - The shifted 56-bit key is **compressed into a 48-bit key**.
    - Compression is achieved by discarding the bits at positions 9, 18, 22, 25, 35, 38, 43, and 54 of the shifted 56-bit key, resulting in $56 - 8 = 48$ bits.
    - The compressed key is then **permuted** according to the following table:

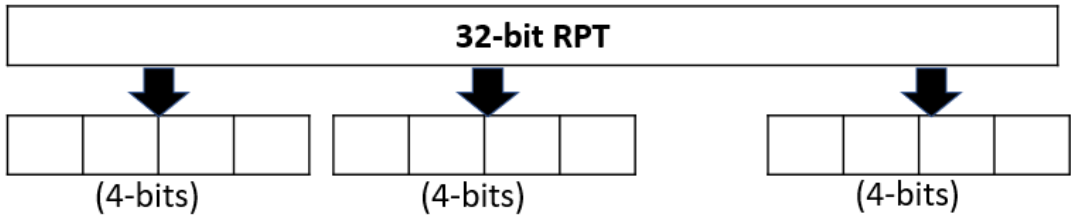
14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

    - The result of this step is **a 48-bit key**.

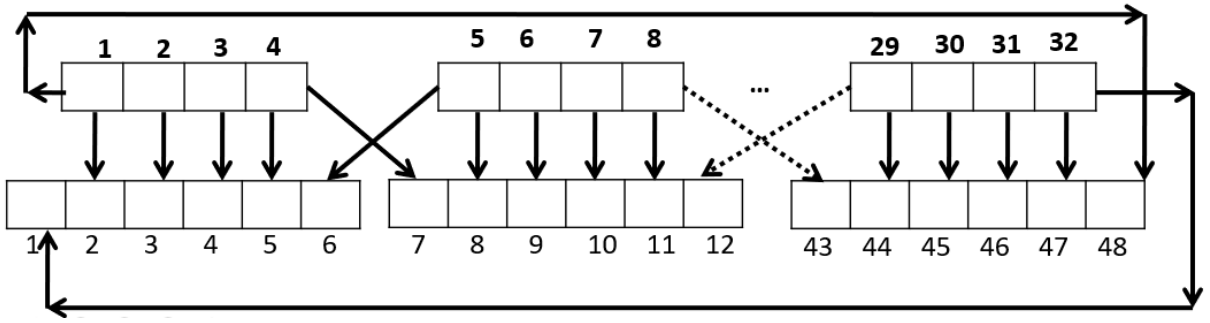
Data Encryption Standard (DES)

- **Step 3:** Each pair of LPT and RPT performs **16 rounds of encryption process**.
  - **Step 3.2: Expansion Permutation of Plain Text and XOR**
    - Since the RPT half obtained from the IP function is 32 bits, it does not match the 48-bit key block derived from the key transformation step (3.1).
    - Therefore, in this step, the RPT half of the plaintext message block is **expanded from 32 bits to 48 bits**.
    - **Expansion Steps:**

A) The 32-bit RPT half is **divided into 8 4-bit blocks**.

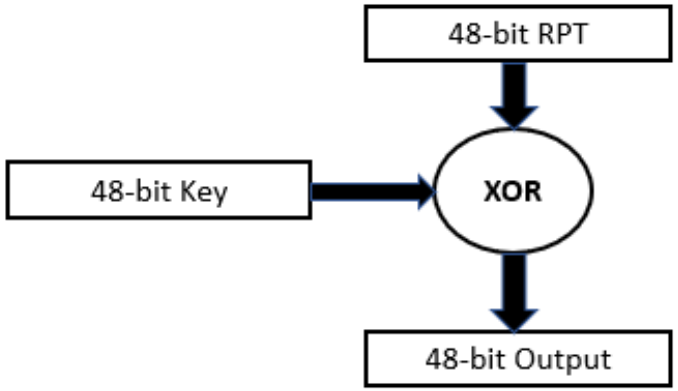


B) Each 4-bit block is **expanded to a 6-bit block**.



Data Encryption Standard (DES)

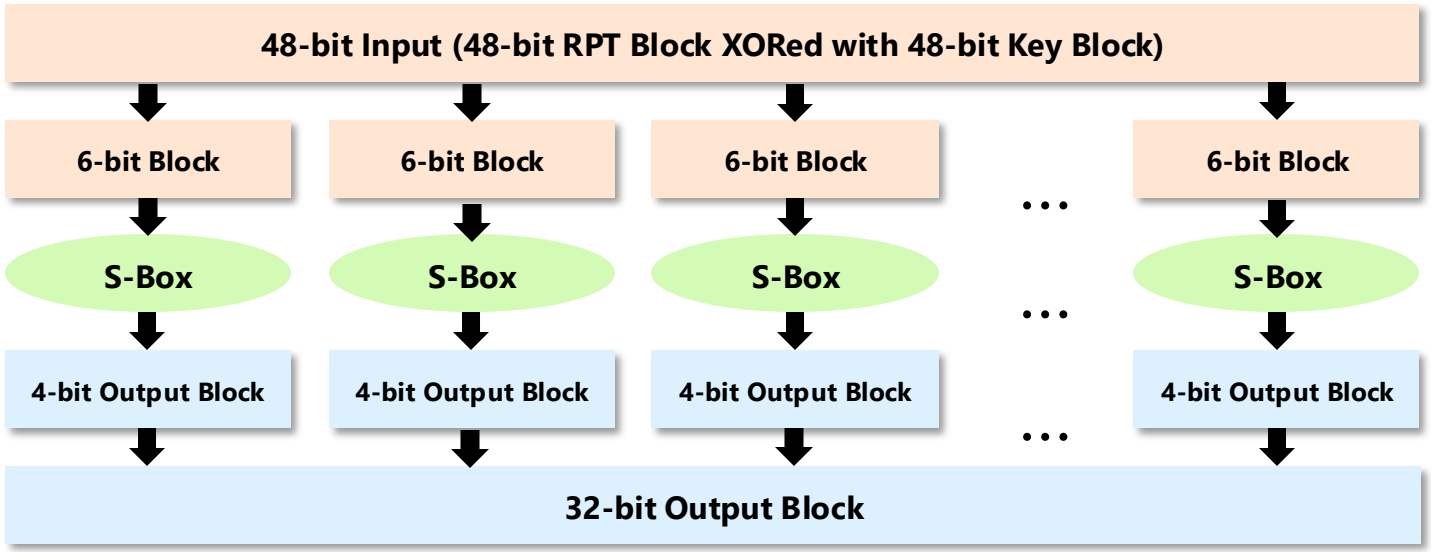
- **Step 3:** Each pair of LPT and RPT performs **16 rounds of encryption process**.
  - **Step 3.2: Expansion Permutation of Plain Text and XOR**
    - Then, the encryption process makes confusion by performing XOR operation between the 48-bit RPT and the 48-bit key blocks.



    - The output of this step is a **48-bit data block**, which will be **the input for the S-Box Substitution** step.

Data Encryption Standard (DES)

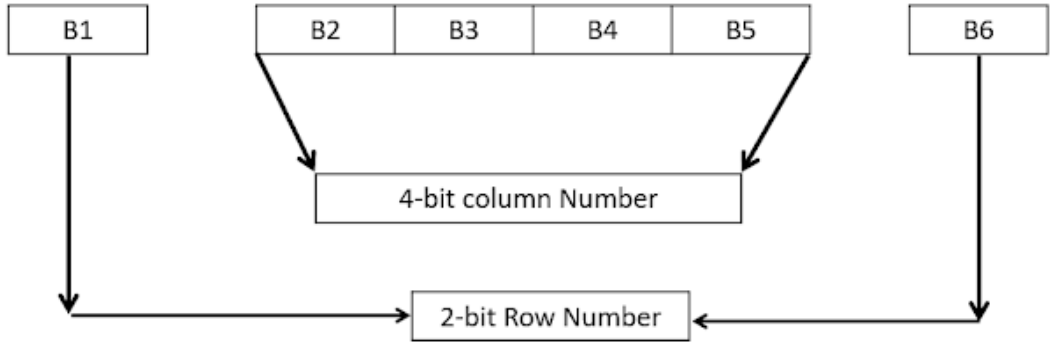
- **Step 3:** Each pair of LPT and RPT performs **16 rounds of encryption process**.
  - **Step 3.3: S-Box Substitution**
    - In this step, the 48-bit output block from the previous step is divided into **eight 6-bit blocks**.
    - Each 6-bit block is then processed through the S-Box function to produce **a 4-bit output block**.



Data Encryption Standard (DES)

- **Step 3:** Each pair of LPT and RPT performs **16 rounds of encryption process**.
  - **Step 3.3: S-Box Substitution**
    - So, this is how the S-Box Substitution function works

S-Box



S_5		Middle 4 bits of input															
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Outer bits	00	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

Example: 011011 → 1001

Data Encryption Standard (DES)

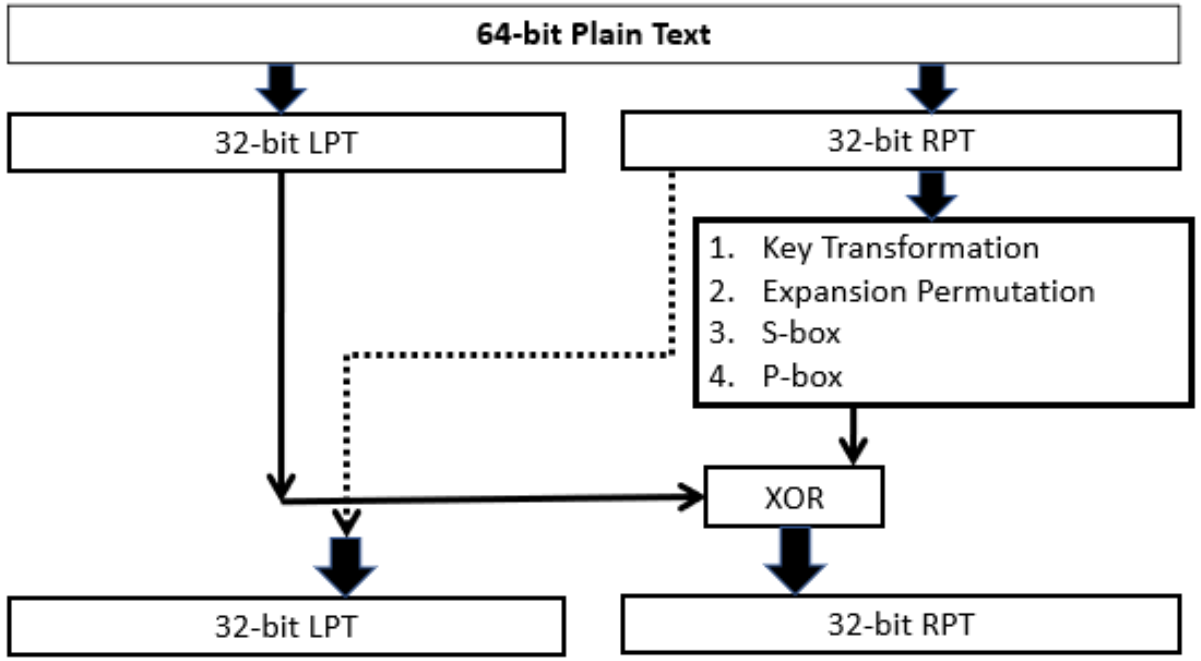
- **Step 3:** Each pair of LPT and RPT performs **16 rounds of encryption process**.
  - **Step 3.4: P-Box Permutation**
    - The 32-bit output block from the S-Box substitution step serves as the input to the P-Box permutation step.
    - The P-Box function **permutes the bit order** of the 32-bit data block according to **the 16×2 permutation table**.
    - **For example:**
      - The 16th bit of the 32-bit data block is moved to the 1st position.
      - The 8th bit of the 32-bit data block is moved to the 17th position.

P – Box Table															
16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

    - The result of this step is **a permuted 32-bit data block**.

Data Encryption Standard (DES)

- **Step 3:** Each pair of LPT and RPT performs **16 rounds of encryption process**.
  - **Step 3.5: XOR and Swap**
    - In this step, an Exclusive-OR (XOR) operation is performed between the output of the previous step and the LPT half.



    - With this, the **first round** of the encryption process is complete. **Fifteen** more rounds remain.

Data Encryption Standard (DES)

- **Step 4:** Resulting LPT and RPT are rejoined, and the **Final Permutation (FP)** is performed on the combined block.
  - At the end of the 16th round of encryption, the LPT and RPT blocks are **concatenated** to form a single data block.
  - This rejoined block is then passed through the FP function once.
  - In this step, the bit order is **permuted** according to **the specified permutation table**.

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

  - **For example:**
    - The 40th bit of the rejoined data block is moved to the 1st position.
    - The 29th bit of the rejoined data block is moved to the 32nd position.
- **Step 5:** A 64-bit cipher text block is successfully generated.

Data Encryption Standard (DES)

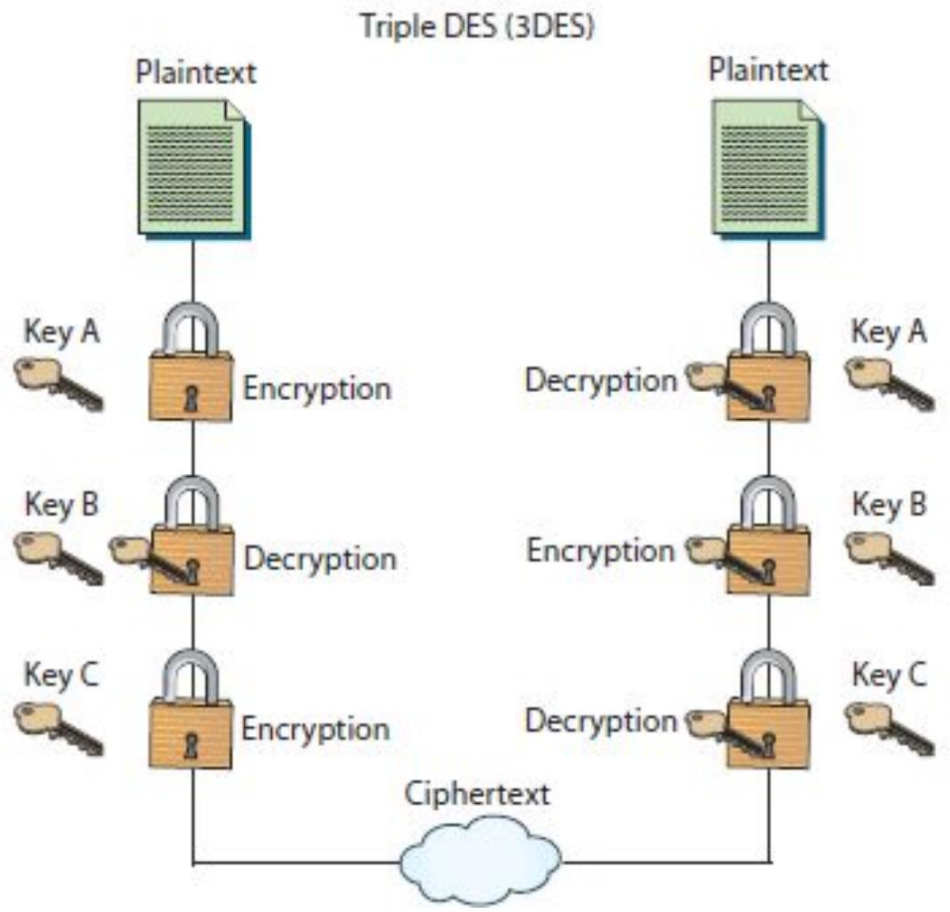
- ***Are these steps too difficult to understand?***
  - Why don't we try to perform those steps on a real example step-by-step?
  - Let's encrypt the following plain text message and key:
    - **Plain text message:** FB14217F50E1A834
    - **Key:** DBC0BB3E9D81EF2A
  - Using this tool: https://simewu.com/des/

Security of DES

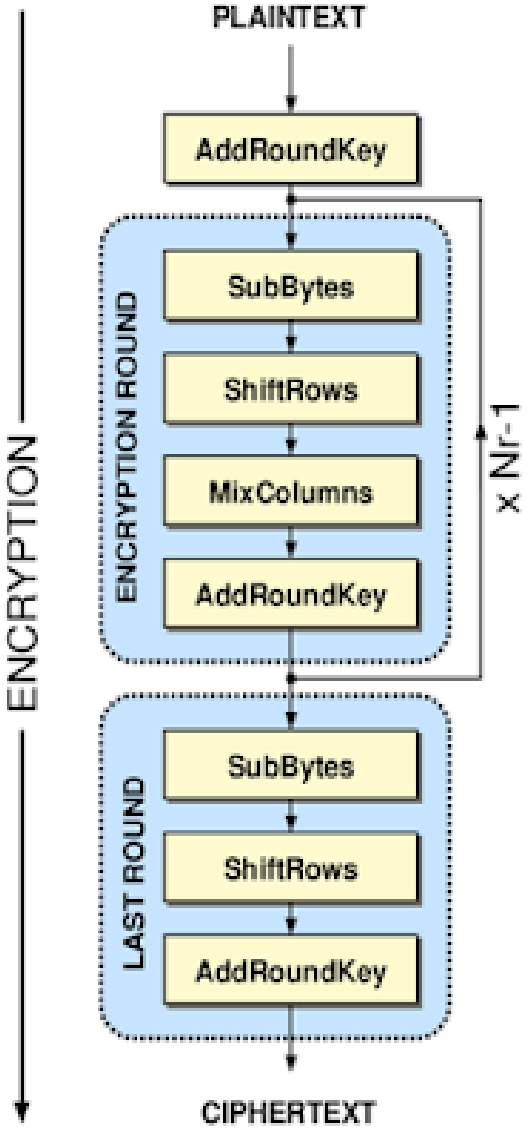
- In 1977, Diffie and Hellman argued that:
  - The 56-bit key length in DES is **too short!**
  - As computers became more powerful, **their performance and speed would eventually exceed the security strength of DES**.
- In later years, researchers used over ***35,000 computers in parallel*** to deduce a DES encryption **key within four months**.
- In 1988, researchers built a special "***DES cracker***" machine capable of finding a DES encryption key in **four days**, at **a cost of under $100,000**.
- In the 1990s, Biham and Shamir introduced a technique called **Differential Cryptanalysis**, which demonstrated that **nearly any modification to the DES algorithm weakened its security**.
  - **For example**, reducing the number of rounds from 16 to 15, altering the expansion rules, or changing the order of operations.

Triple Data Encryption Standard (3DES)

- **Triple DES (3DES)** is a variant of DES.
  - 3DES uses **two or three encryption keys**.
  - It applies multiple encryption, meaning that the plaintext message is processed **through the DES algorithm two or three times**.
  - Although 3DES is stronger than DES, it **inherits the same fundamental weaknesses**.
- 3DES was **first published in 1995** and was **officially deprecated in 2017**.



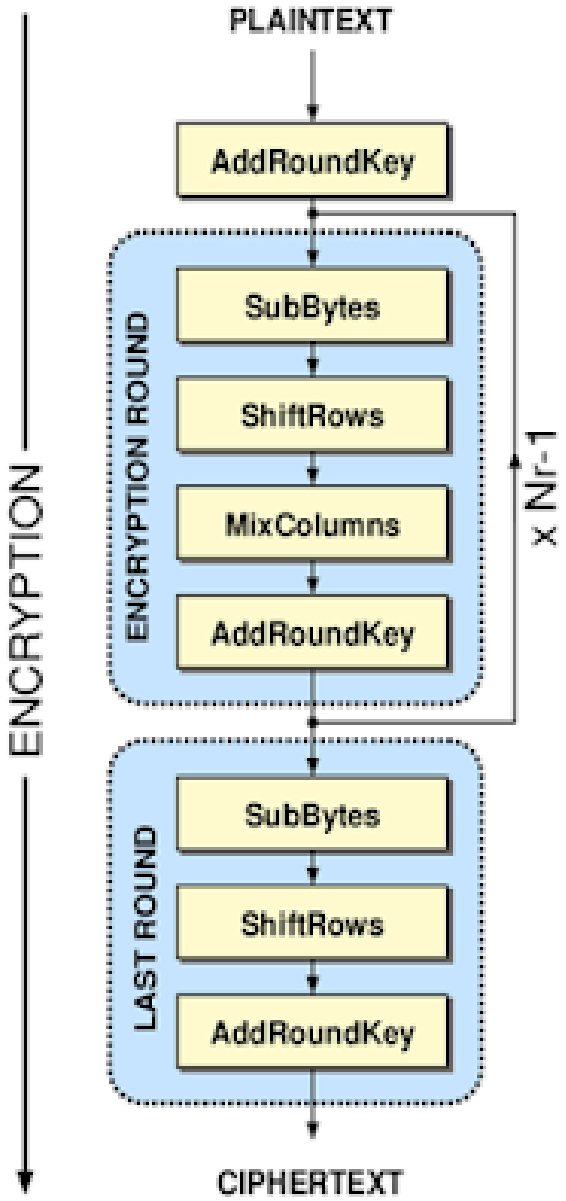
Advanced Encryption Standard (AES)



- **Advanced Encryption Standard (AES)** is the most recent and widely used strong symmetric key encryption algorithm.
- **AES** was developed by Belgian cryptographers *Joan Daemen* and *Vincent Rijmen*, whose algorithm "**Rijndael**" won a competition held by the U.S. government to replace the aging DES standard.
  - Sometimes, AES is also referred to as the "**Rijndael Cipher**."
- **AES** performs encryption in multiple rounds, with **the number of rounds depending on the key size**:
  - 10 rounds for a 128-bit key
  - 12 rounds for a 192-bit key
  - 14 rounds for a 256-bit key
- **AES** operates on **a 128-bit input block** (equivalent to 16 characters) and produces **a 128-bit ciphertext block**.

Advanced Encryption Standard (AES)

- There are 5 steps in the AES encryption algorithm:
  1. **Plaintext Encoding:** Each character in the plaintext block is converted into its hexadecimal representation and arranged into a 4×4 matrix known as the **state matrix**.
  2. **Substitute Bytes:** Each byte in the matrix is replaced with another byte using a fixed **substitution box (S-box)** to introduce **confusion**.
  3. **Shift Rows:** Each row of the matrix is **cyclically shifted** to the left by a certain number of positions to provide **diffusion**.
  4. **Mix Columns:** Each column in the matrix is **mathematically transformed** using matrix multiplication in a Galois Field to further spread the data, enhancing **diffusion**.
  5. **Add Round Key:** A round-specific key is **XOR-ed** with the state matrix to add confusion and tie the encryption to the specific key being used.



AES Step 1: Plaintext Encoding (Text to 4 x 4 Matrix)

- We encode each character in a plaintext block into hexadecimal and organize them into a 4×4 matrix, known as the state array, as follows:
  1. Prepare a plaintext block **with up to 16 characters**. If the block has fewer than 16 characters, it should be padded appropriately.
  2. Convert each character into its corresponding **hexadecimal representation** using the ASCII table.
  3. Arrange the 16 hex values into **a 4 × 4 matrix** in column-major order (i.e., filling the matrix column by column).

Example: Plaintext: "AES uses matrix operation"

Plaintext Block: A E S U S E S M A T R I X O P E

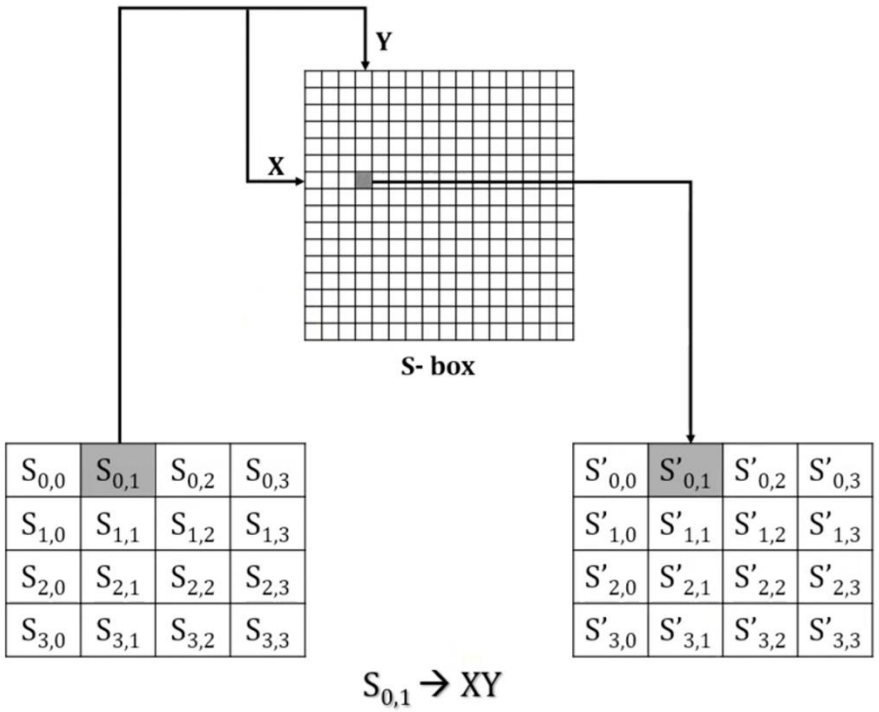
Hexadecimal: 00 04 12 14 12 04 12 0C 00 13 11 08 17 0E 0F 04

$$\begin{bmatrix} 00 & 12 & 00 & 17 \\ 04 & 04 & 13 & 0E \\ 12 & 12 & 11 & 0F \\ 14 & 0C & 08 & 04 \end{bmatrix}$$

	DEC	HEX		DEC	HEX
A	00	00	N	13	0D
B	01	01	O	14	0E
C	02	02	P	15	0F
D	03	03	Q	16	10
E	04	04	R	17	11
F	05	05	S	18	12
G	06	06	T	19	13
H	07	07	U	20	14
I	08	08	V	21	15
J	09	09	W	22	16
K	10	0A	X	23	17
L	11	0B	Y	24	18
M	12	0C	Z	25	19

AES Step 2: Substitute Bytes

- We substitute each element in the state array (4×4 matrix) with its corresponding value from the **Substitution Box (S-Box)**, using the element's hexadecimal value as the lookup index.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

AES Step 2: Substitute Bytes

- **Example:**

$$StateArray = \begin{bmatrix} 00 & 12 & 00 & 17 \\ 04 & 04 & 13 & 0E \\ 12 & 12 & 11 & 0F \\ 14 & 0C & 08 & 04 \end{bmatrix}$$

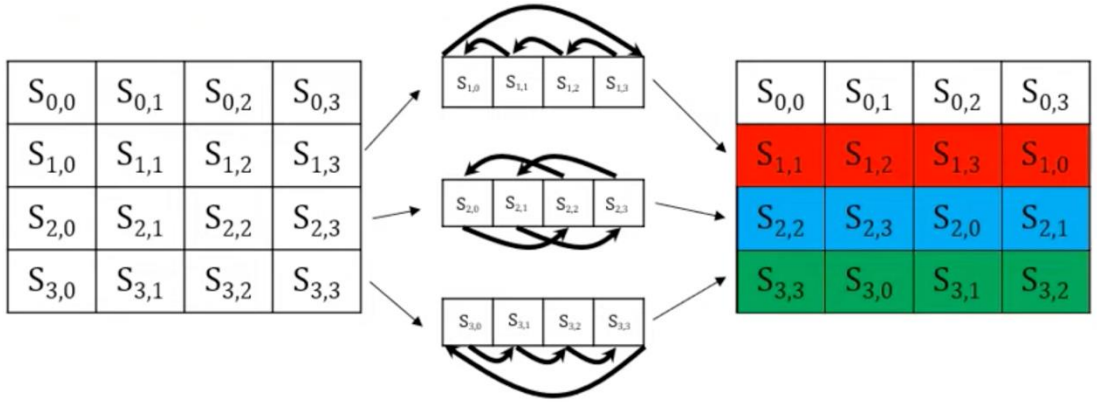
$$StateArray = \begin{bmatrix} 63 & & & \\ & & & \\ & & & \\ FA & & & \end{bmatrix}$$

$$StateArray = \begin{bmatrix} 63 & C9 & 63 & F0 \\ F2 & F2 & 7D & AB \\ C9 & C9 & 82 & 76 \\ FA & FE & 30 & F2 \end{bmatrix}$$

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

AES Step 3: Shift Rows

- We shift the elements in each row of the state matrix using **circular left shifts with offsets based on the row number**:
  - The **first row** is not shifted (0 positions).
  - The **second row** is circularly left-shifted by 1 position.
  - The **third row** is circularly left-shifted by 2 positions.
  - The **fourth row** is circularly left-shifted by 3 positions (equivalent to a circular right shift by 1 position).



$$StateArray = \begin{bmatrix} 63 & C9 & 63 & F0 \\ F2 & F2 & 7D & AB \\ C9 & C9 & 82 & 76 \\ FA & FE & 30 & F2 \end{bmatrix}$$

$$StateArray = \begin{bmatrix} 63 & C9 & 63 & F0 \\ F2 & 7D & AB & F2 \\ 82 & 76 & C9 & C9 \\ F2 & FA & FE & 30 \end{bmatrix}$$

AES Step 4: Mix Columns

- We multiply **the shifted state array** with a predefined matrix using a dot-product operation, similar to standard matrix multiplication.

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

Predefine Matrix

*

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

State Array

=

$S'_{0,0}$	$S'_{0,1}$	$S'_{0,2}$	$S'_{0,3}$
$S'_{1,0}$	$S'_{1,1}$	$S'_{1,2}$	$S'_{1,3}$
$S'_{2,0}$	$S'_{2,1}$	$S'_{2,2}$	$S'_{2,3}$
$S'_{3,0}$	$S'_{3,1}$	$S'_{3,2}$	$S'_{3,3}$

New State Array

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} * \begin{bmatrix} 63 & C9 & 63 & F0 \\ F2 & 7D & AB & F2 \\ 82 & 76 & C9 & C9 \\ F2 & FA & FE & 30 \end{bmatrix} = \begin{bmatrix} 63 * 02 + \\ F2 * 03 + \\ 82 * 01 + \\ F2 * 01. \end{bmatrix}$$

AES Step 4: Mix Columns

- We multiply **the shifted state array** with a predefined matrix using a dot-product operation, similar to standard matrix multiplication.

Example: $02 * 63$

$$X^7 + X^6 + X^5 + X^4 + X^3 + X^2 + X^1 + X^0$$

$$0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0$$

$$02 = 0\,0\,0\,0\,0\,0\,1\,0 = X$$

$$63 = 0\,1\,1\,0\,0\,0\,1\,1 = X^6 + X^5 + X + 1$$

$$X^7 + X^6 + X^5 + X^4 + X^3 + X^2 + X^1 + X^0$$

$$0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1$$

$$02 * 63 = X * (X^6 + X^5 + X + 1) = X^7 + X^6 + X^2 + X = 1100\,0110$$

If the multiplication go beyond X^7, we will use irreducible polynomial theorem, $GF(2^3)$

$$X^8 = X^4 + X^3 + X + 1$$

AES Step 4: Mix Columns

- We multiply **the shifted state array** with a predefined matrix using a dot-product operation, similar to standard matrix multiplication.

$$\begin{bmatrix} 63 * 02 + \\ F2 * 03 + \\ 82 * 01 + \\ F2 * 01. \end{bmatrix}$$

$$\begin{bmatrix} DD & ?? & ?? & ?? \\ ?? & ?? & ?? & ?? \\ ?? & ?? & ?? & ?? \\ ?? & ?? & ?? & ?? \end{bmatrix}$$

$$63 * 02 = 1100\ 0110$$
$$F2 * 03 = 0000\ 1101 \quad \oplus$$
$$82 * 01 = 1000\ 0010$$
$$F2 * 01 = 1111\ 0010$$

$$1011\ 1011$$
$$= DD$$

XOR Operation:

- Odd times 1's = 1
- Even times 1's = 0

$$F2 = 1111\ 0010 = X^7 + X^6 + X^5 + X^4 + X$$
$$03 = 0000\ 0011 = X + 1$$

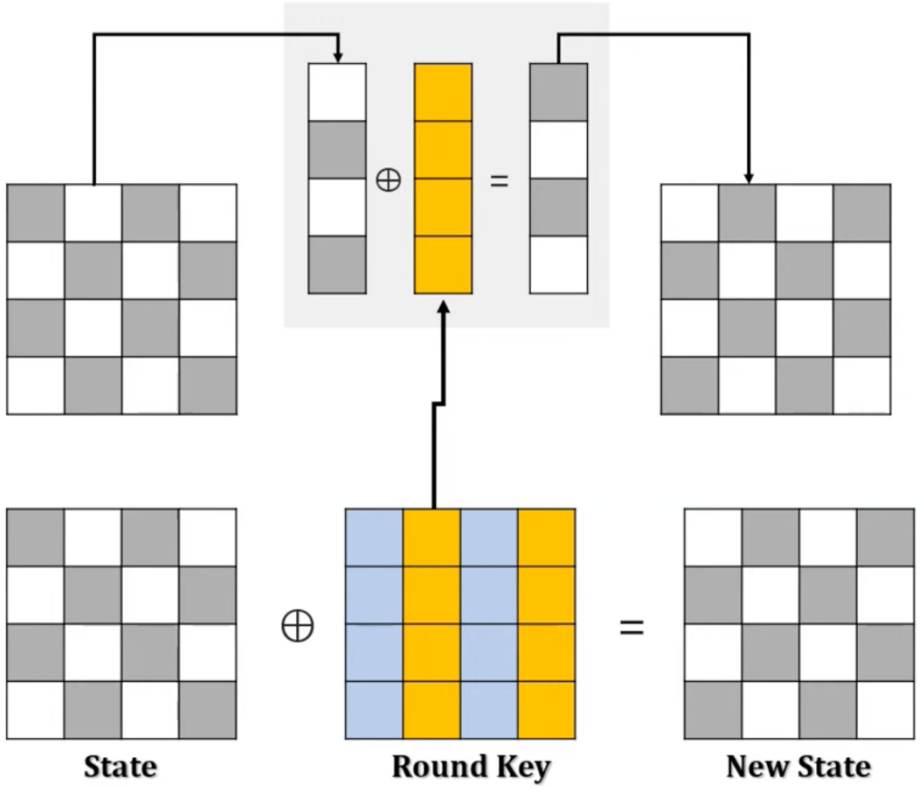
$$F2 * 03 = (X + 1)(X^7 + X^6 + X^5 + X^4 + X)$$
$$= X^8 + \cancel{2X^7 + 2X^6 + 2X^5} + X^4 + X^2 + X$$
$$= \cancel{X^4} + X^3 + \cancel{X} + 1 + \cancel{X^4} + X^2 + \cancel{X}$$
$$= X^3 + X^2 + 1 = 0000\ 1101$$

Hex Value Calculator:
https://www.calculator.net/hex-calculator.html

Binary XOR Calculator:
https://www.compscilib.com/calculate/binaryxor

AES Step 5: Add Round Key

- Perform an XOR operation between the round key and the state array.
  - The XOR operation is applied **column by column**.



47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

$\oplus$

AC	19	28	57
77	FA	D1	5C
66	DC	29	00
F3	21	41	6A

=

EB			
40			
F2			
1E			

$47 \oplus AC = ?$
$47 = 0100\ 0111$
$AC = 1010\ 1100$
$= 1110\ 1011$
$E\quad B = \{EB\}$

$37 \oplus 77 = ?$
$37 = 0011\ 0111$
$77 = 0111\ 0111$
$= 0100\ 0000$
$4\quad 0 = \{40\}$

$94 \oplus 66 = ?$
$94 = 1001\ 0100$
$66 = 0110\ 0110$
$= 1111\ 0010$
$F\quad 2 = \{F2\}$

$ED \oplus F3 = ?$
$ED = 1110\ 1101$
$F3 = 1111\ 0011$
$= 0001\ 1110$
$1\quad E = \{1E\}$

Advanced Encryption Standard (AES)

Congratulation!

- We just finished one round of AES encryption process.

- AES performs encryption in multiple rounds (similar to DES), with **the number of rounds depending on the key size**.
  - 10 rounds for 128-bit keys
  - 12 rounds for 192-bit keys
  - 14 rounds for 256-bit keys

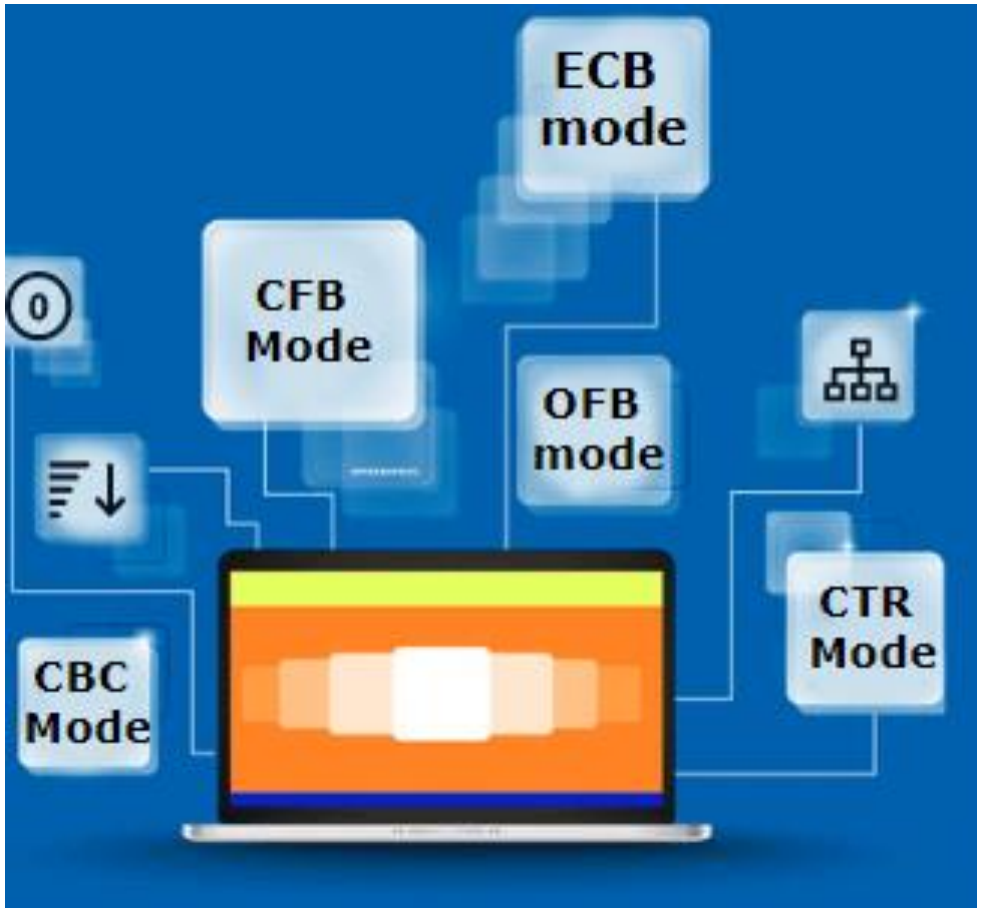
Block Ciphers with Long Plaintext

- Up to now, we know that block ciphers define a specific plaintext block size.
  - For plaintext messages **shorter** than the block size, block ciphers typically pad them with additional characters (often random or predefined).
  - For plaintext messages **longer** than the block size, block ciphers split them into multiple blocks.
    Normally, each block is treated **independently** during the encryption and decryption processes.

- **Question:** Would it be more secure if we introduced dependencies among the blocks?
  - **The answer is yes!**
  - If we can pass information from the previous block to the next one, it strengthens the encryption process.
  - This concept is known as the "**modes of operation**" of block ciphers.

Modes of Operation for Block Ciphers

- There are five modes of operation that can be applied to block ciphers:
  - **Electronic Code Block (ECB) mode**
  - **Code Block Chaining (CBC) mode**
  - **Cipher Feedback (CFB) mode**
  - **Output Feedback (OFB) mode**
  - **Counter (CTR) mode**

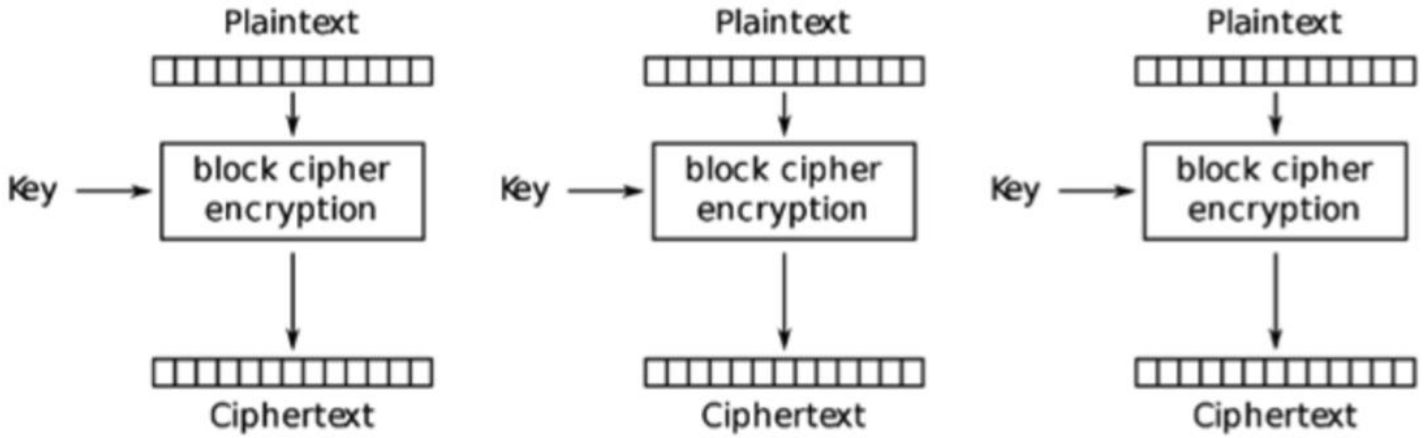


Cautions:

- If you are using a cryptographic library and it asks you to select the modes of operation, please consider **do not use this library**. It should not ask users to do this!
- The right way to use AES is to select **the Authenticated Encryption with Associated Data (AEAD) mode** where the cryptographer has already chosen the right mode of operation for you.

Modes of Operation for Block Ciphers

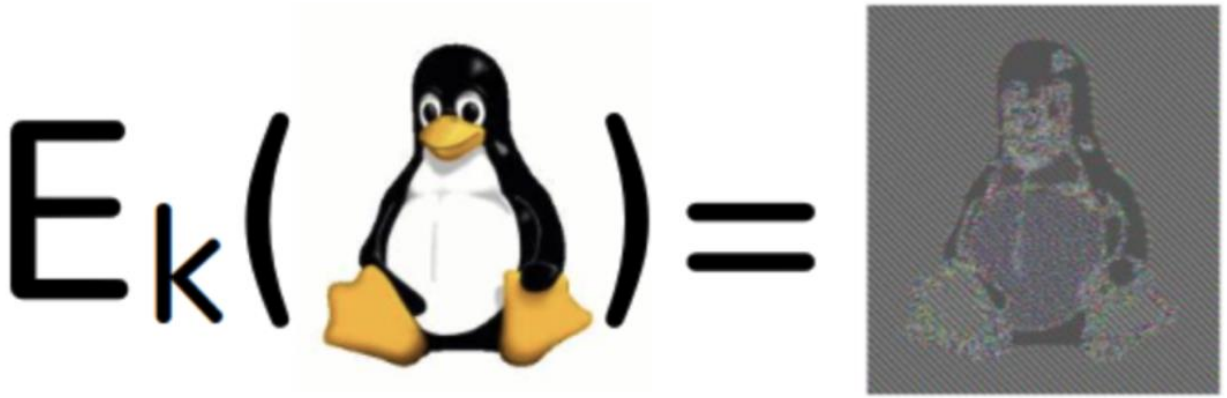
- **Electronic Code Block (ECB) Mode**



- This is the simplest way you might think of when you are using a block cipher.
- With this mode, you just break your (padded) plaintext message into blocks and operate the encryption process separately. The ciphertext will be concentrated from different blocks.
- *This is terribly insecure because the same input block produces the same output block!*

Modes of Operation for Block Ciphers

- **Electronic Code Block (ECB) Mode**

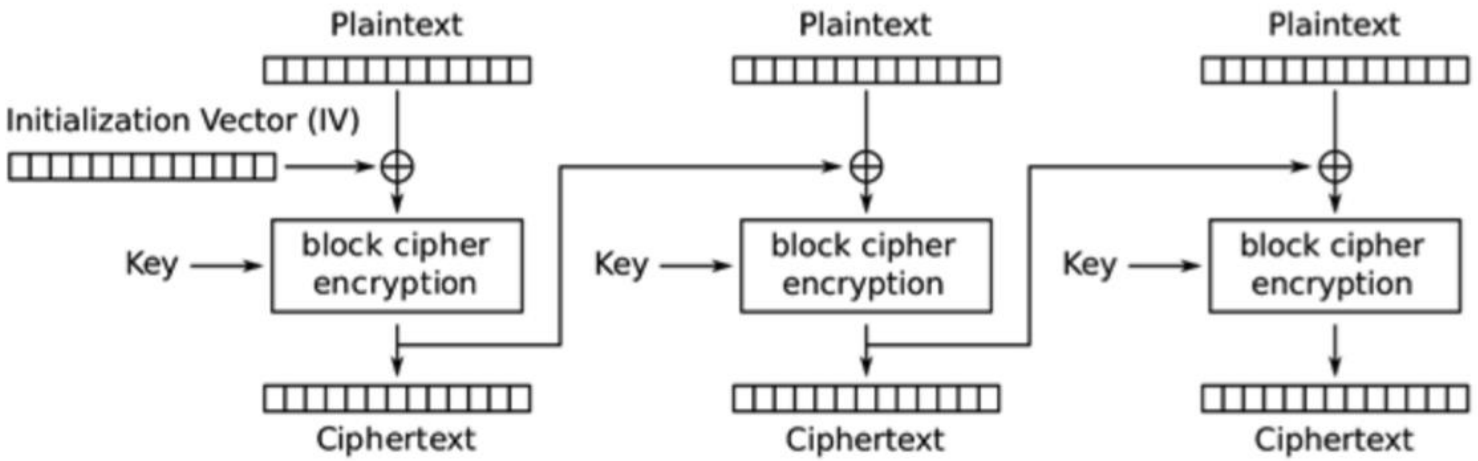


- The most famous illustration of the insecurity of the ECB mode is the "**ECB Penguin**".
- This is a bitmap of the Tux penguin that was encrypted using ECB mode. You can still see the penguin.

Remember that ECB mode lets you see the penguin, so you should never use it.

Modes of Operation for Block Ciphers

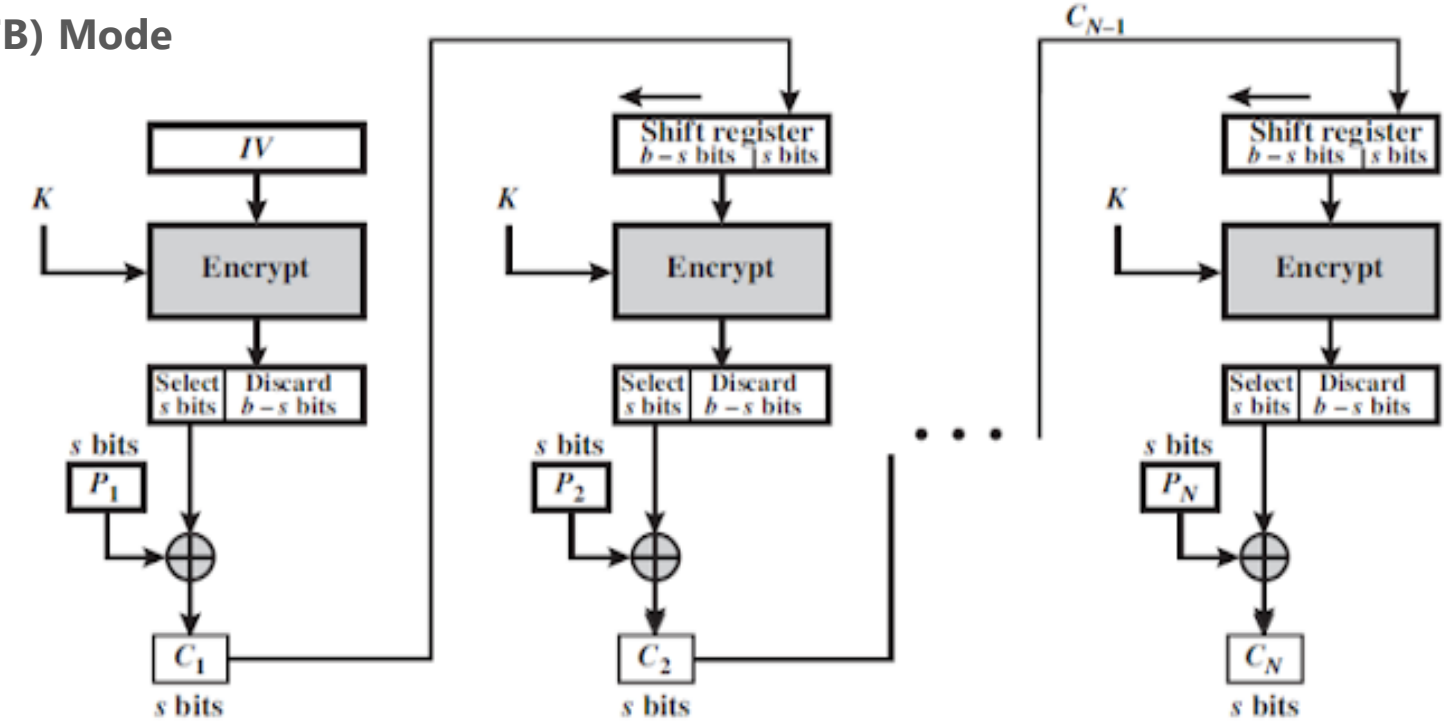
- **Code Block Chaining (CBC) Mode**



- CBC addresses the weakness of the ECB mode by adding ***randomization*** into the ciphertext.
- There is a value called **Initialization Vector (IV)** that has been used to randomize the input in the first block encryption. Then, the first ciphertext will be **XORed** with the second plaintext input block.
- ***Unfortunately, it is difficult to implement properly, and it is found to be vulnerable to padding oracle attacks.***

Modes of Operation for Block Ciphers

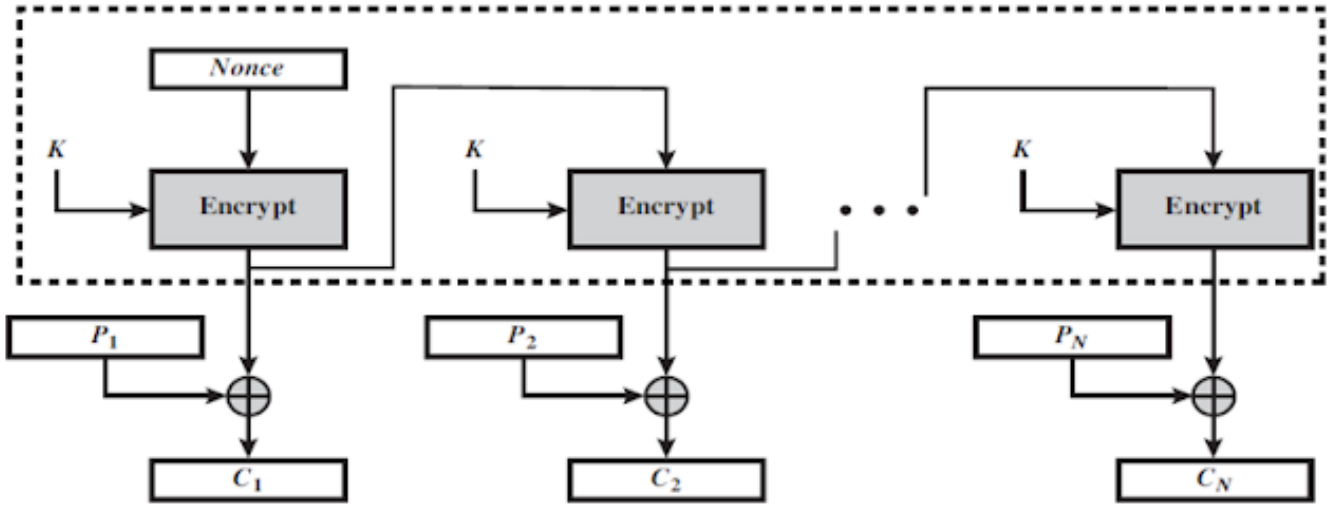
- **Cipher Feedback (CFB) Mode**



- The CFB mode uses the concept of block ciphers, but it is actually acting as a stream cipher.
- Data is encrypted in a smaller size of block, such as 8-bit data block rather than a predefined block size of 64 bits.
- This mode is mostly used in the **key stream generator**.

Modes of Operation for Block Ciphers

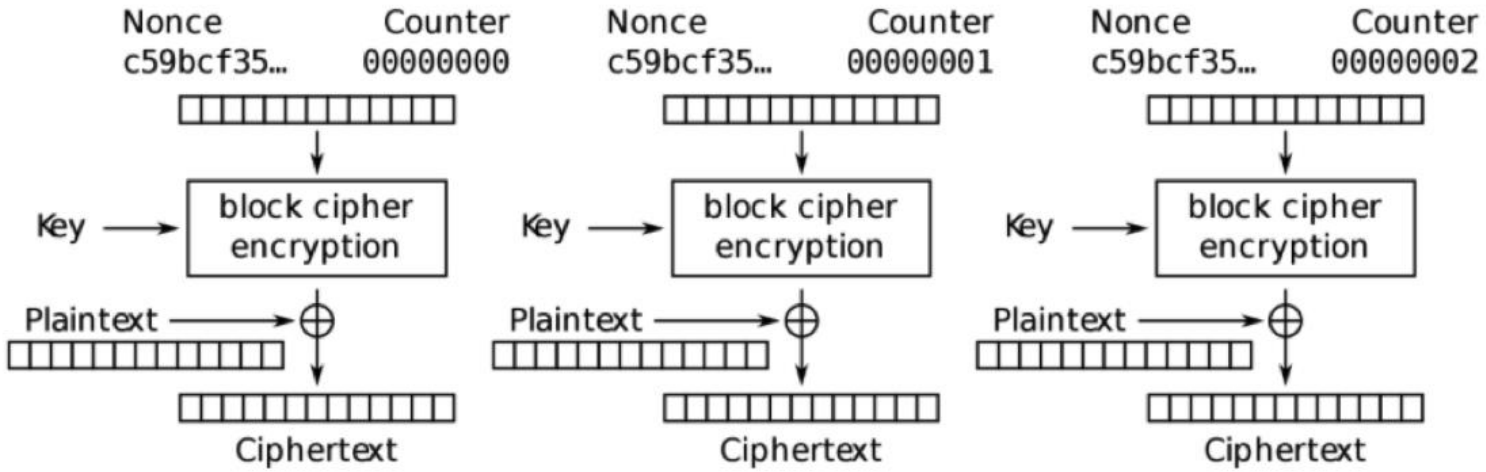
- **Output Feedback (OFB) Mode**



- The OFB mode is similar in structure to that of CFB mode. The difference is that output of encryption process O_1 is directly placed in the next stage of shift register without XOR operation instead of generating a ciphertext C_1.
- If there is **a small error in individual bits**, it remains an error in individual bits which **do not corrupt the whole encrypted message** which is ***the biggest advantage over all other modes***.

Modes of Operation for Block Ciphers

- **Counter (CTR) Mode**



- In CTR mode, the idea is to make **a stream of outputs** that look like a stream cipher.
- You can pick a random starting point and encrypt the initial counter value with the block cipher to obtain the first block. Then, the second block can be obtained from the encryption of counter+1, and so on.
- You **XOR** the message with **this pseudorandom output stream** to obtain your ciphertext.

Modes of Operation for Block Ciphers

- **Applications/Real-World Usage of Modes of Operation for Block Ciphers**

Electronic Codebook (ECB) Mode	• Secure transmission of a single value like password and encryption key.
Cipher Block Chaining (CBC) Mode	• General-purpose block-oriented transmission. i.e., Authentication
Cipher Feedback (CFB) Mode	• General-purpose stream-oriented transmission. i.e., Authentication
Output Feedback (OFB) Mode	• Transmission over noisy channel. i.e., satellite communication
Counter (CTR) mode	• Block oriented transmission, where application needs high speed.

Modes of Operation for Block Ciphers

- **How to Implement Block Ciphers in Real Python Applications**

Key Generation and Encryption

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

data = b'secret data'

key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_EAX)
ciphertext, tag = cipher.encrypt_and_digest(data)
nonce = cipher.nonce
```

Decryption

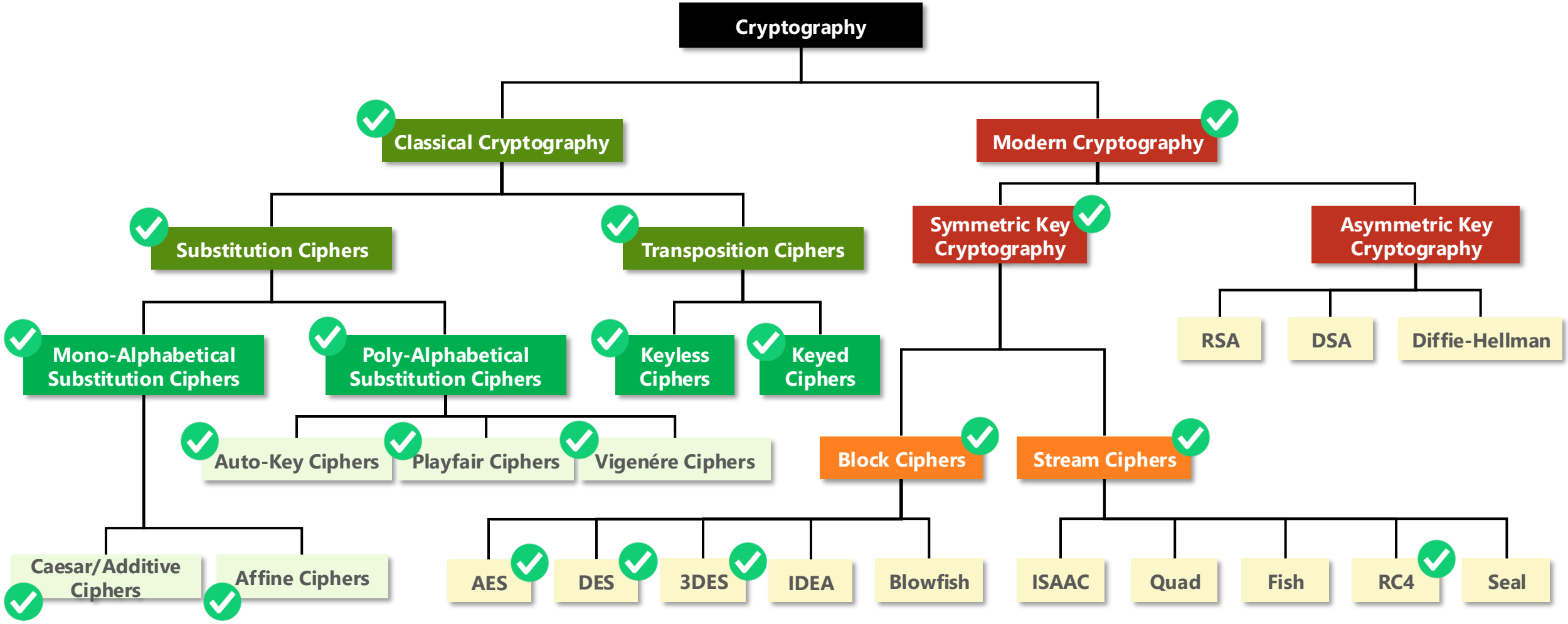
```
cipher = AES.new(key, AES.MODE_EAX, nonce)
data = cipher.decrypt_and_verify(ciphertext, tag)
```

Stream Ciphers Versus Block Ciphers

Stream Ciphers	Block Ciphers
✅ Faster than block ciphers in operation.	❌ Require more memory to process.
❌ More difficult to implement it correctly.	✅ Stronger.
❌ Low Diffusion.	✅ High Diffusion.
❌ Susceptible to Insertions and/or modifications	✅ Resistant to Insertions/Modifications.
✅ Low error propagations.	❌ Susceptible to error propagations.
❌ Cannot provide integrity or authentication protections.	✅ Can be used for authentication and integrity verification.
Common Algorithm: A5, RC4	**Common Algorithms:** 3DES, AES

Question: ***Which cipher will you use in your secure software application?***

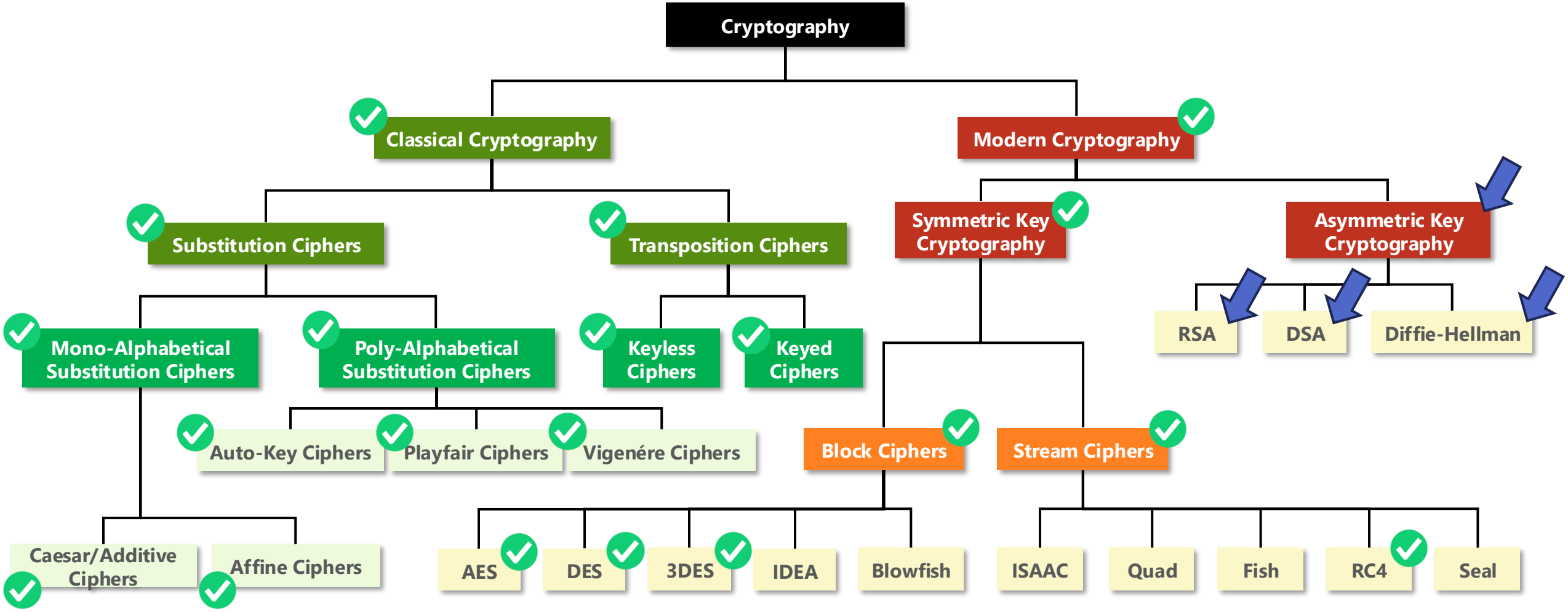
Classification of Cryptographic Techniques



Section 4.2:

Asymmetric Key Cryptography

Classification of Cryptographic Techniques



An Introduction of Asymmetric Key Cryptography

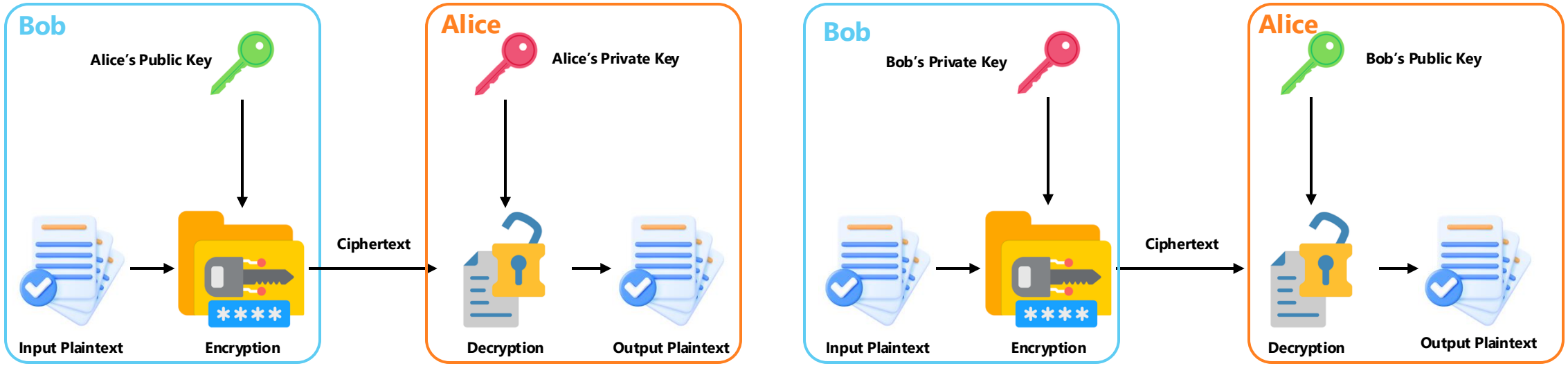
- As discussed in the previous lecture, **symmetric key cryptography** is a cryptographic technique in which both the sender and receiver use the same secret key for encryption and decryption.

- In contrast, **asymmetric key cryptography**, also known as **public key cryptography**, uses a pair of keys – a public key for encryption and a private key for decryption.
- The key differences between symmetric and asymmetric key cryptography are summarized in the table below.

Asymmetric Key Cryptography	Symmetric Key Cryptography
Plaintext (P)	Plaintext (P)
Ciphertext (C)	Ciphertext (C)
Public Key (PK)	**Encryption Key (K)**
Private Key or Secret Key (SK)	
Encryption Algorithm ($E(\cdot)$ or $Encrypt(\cdot)$)	Encryption Algorithm ($E(\cdot)$ or $Encrypt(\cdot)$)
Decryption Algorithm ($D(\cdot)$ or $Decrypt(\cdot)$)	Decryption Algorithm ($D(\cdot)$ or $Decrypt(\cdot)$)

Use Case Scenarios for Asymmetric Key Cryptography

- **A public key (PK)** is a cryptographic key that *can be openly shared or published*. It does not provide sufficient capability on its own to break the encryption process.
- **A private key or secret key (SK)** must be *kept confidential by its owner*. If this key is exposed, the security of the encryption process is compromised.
- There are **two primary use cases** for asymmetric key cryptography:



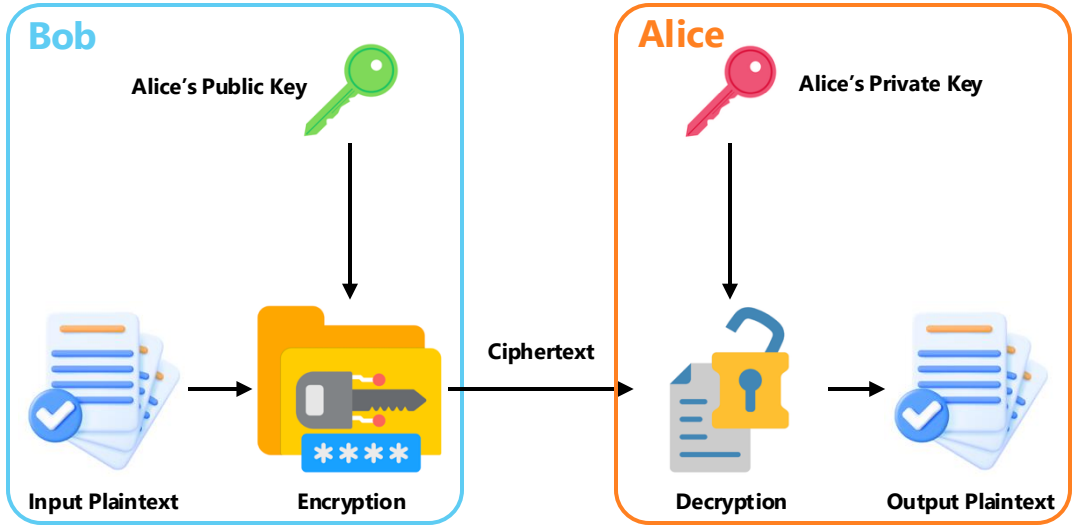
Confidentiality: ***PK*** for encryption, ***SK*** for decryption

Integrity: ***SK*** for encryption, ***PK*** for decryption

Use Case Scenarios for Asymmetric Key Cryptography

- **Scenario 1: Encryption with A Public Key**
  - The process can be represented as:
  - $\boldsymbol{C} = Encrypt(PK_{Alice}, \boldsymbol{P}_{Bob \rightarrow Alice})$
  - $\boldsymbol{P} = Decrypt(SK_{Alice}, \boldsymbol{C}_{Bob \rightarrow Alice})$

    where:
    - PK_{Alice} is Alice's public key,
    - SK_{Alice} is Alice's private or secret key,
    - $\boldsymbol{P}_{Bob \rightarrow Alice}$ is a plaintext message that Bob intended to send to Alice, and
    - $\boldsymbol{C}_{Bob \rightarrow Alice}$ is a ciphertext that Bobs actually transmits to Alice.



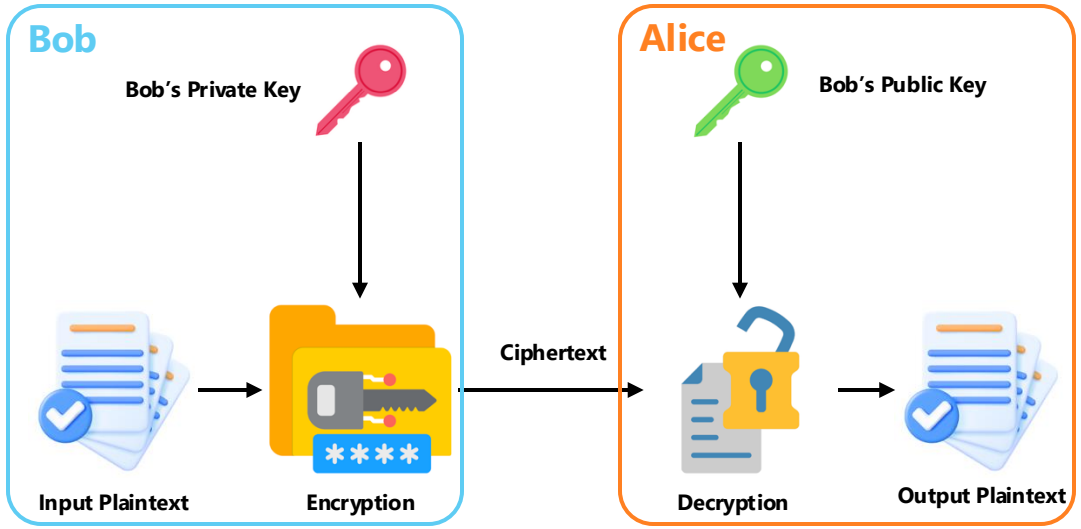
Confidentiality: ***PK*** for encryption, ***SK*** for decryption

It ensures confidentiality because only an intended recipient can decrypt and read the message.

Use Case Scenarios for Asymmetric Key Cryptography

- **Scenario 2: Encryption with A Private Key**
  - The process can be represented as:
  - $\boldsymbol{C} = Encrypt(SK_{Bob}, \boldsymbol{P}_{Bob \rightarrow Alice})$
  - $\boldsymbol{P} = Decrypt(PK_{Bob}, \boldsymbol{C}_{Bob \rightarrow Alice})$

    where:
    - PK_{Bob} is Bob's public key,
    - SK_{Bob} is Bob's private or secret key,
    - $\boldsymbol{P}_{Bob \rightarrow Alice}$ is a plaintext message that Bob intended to send to Alice, and
    - $\boldsymbol{C}_{Bob \rightarrow Alice}$ is a ciphertext that Bobs actually transmits to Alice.



Integrity: ***SK*** for encryption, ***PK*** for decryption

It ensures integrity in term of message authentication, because the recipient knows that the message is sent by the public key's owner.

Mathematical Fundamentals for Asymmetric Key Cryptography

- **Prime Numbers**
  - A prime number (or simply, a prime) is a natural number **greater than 1** that **is not the product of two smaller natural numbers**.
  - A natural number greater than 1 that **is not prime** is called **a composite number**.
  - **For example**, 2, 3, 5, 7, 11, 13, 17, 19 are prime numbers.
- **Relative Prime**
  - If *the only common factor of two numbers a and b is 1* (i.e., $GCD(a, b) = 1$), then a and b are called **relatively prime numbers**.
  - In this case, the pair (a, b) is said to be **a relatively prime pair**.
  - **These numbers do not need to be prime themselves**; two composite numbers can also be relatively prime.
    For example, 9 and 10 are relatively prime, but they are not prime numbers.
  - Relatively prime numbers are also referred to as **mutually prime** or **coprime numbers**.

Mathematical Fundamentals for Asymmetric Key Cryptography

- **Primitive Root Modulo n**
  - A number g is called a primitive root modulo n if every integer that is coprime to n is congruent to some power of g **modulo n**.
  - A primitive root modulo a prime p is an integer g such that the powers of g modulo p generate all the nonzero residues modulo p. In other words, **g mod p has multiplicative order $p - 1$**.

Example 1: Is 3 a primitive root modulo 7?

$$3^1 = 3\ mod\ 7 \equiv 3$$
$$3^2 = 9\ mod\ 7 \equiv 2$$
$$3^3 = 27\ mod\ 7 \equiv 6$$
$$3^4 = 81\ mod\ 7 \equiv 4$$
$$3^5 = 243\ mod\ 7 \equiv 5$$
$$3^6 = 729\ mod\ 7 \equiv 1$$

Hence, 3 is a primitive root modulo 7.

Example 2: Is 3 a primitive root **modulo 11**?

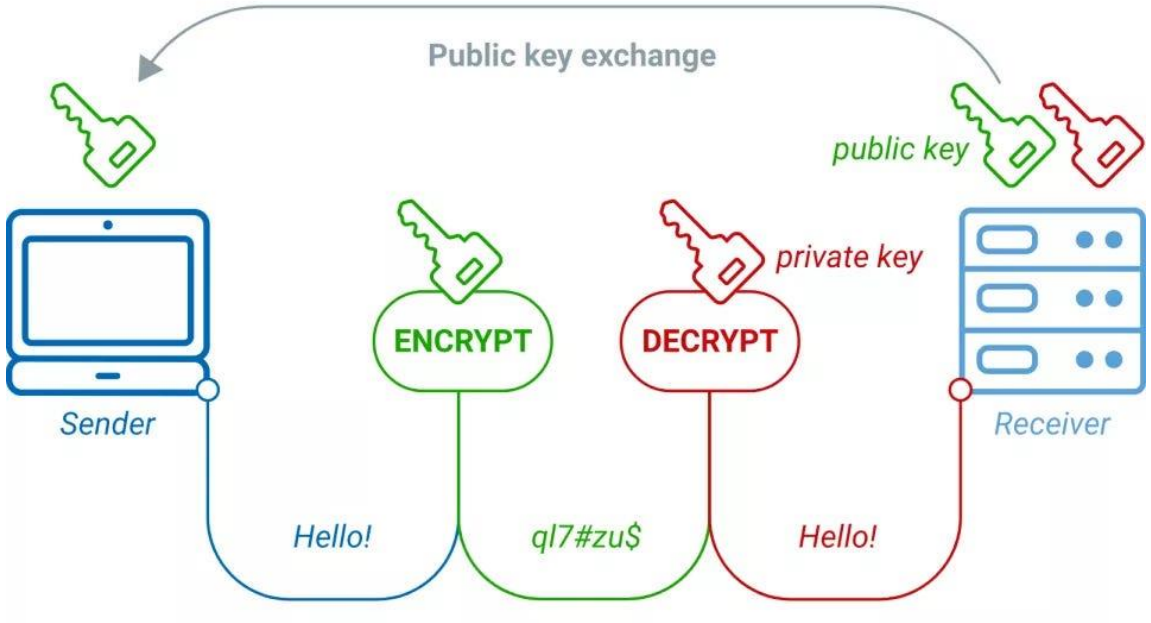
$$3^1 = 3\ mod\ 11 \equiv 3$$
$$3^2 = 9\ mod\ 11 \equiv 9$$
$$3^3 = 27\ mod\ 11 \equiv 5$$
$$3^4 = 81\ mod\ 11 \equiv 4$$
$$3^5 = 243\ mod\ 11 \equiv 1$$
$$3^6 = 729\ mod\ 11 \equiv \mathbf{3}$$

Multiplicative order of $3\ mod\ 11$ is **5**

Hence, 3 is **not** a primitive root modulo **11**.

Asymmatric Key Cryptographic Algorithms

- There are significantly fewer asymmetric key encryption algorithms compared to symmetric ones. This is because asymmetric encryption relies on **more complex mathematical foundations**, making its design and implementation more challenging.
- Two widely recognized asymmetric key encryption algorithms are:
  - **Diffie-Hellman Key Exchange Mechanism**
  - **RSA Algorithm**

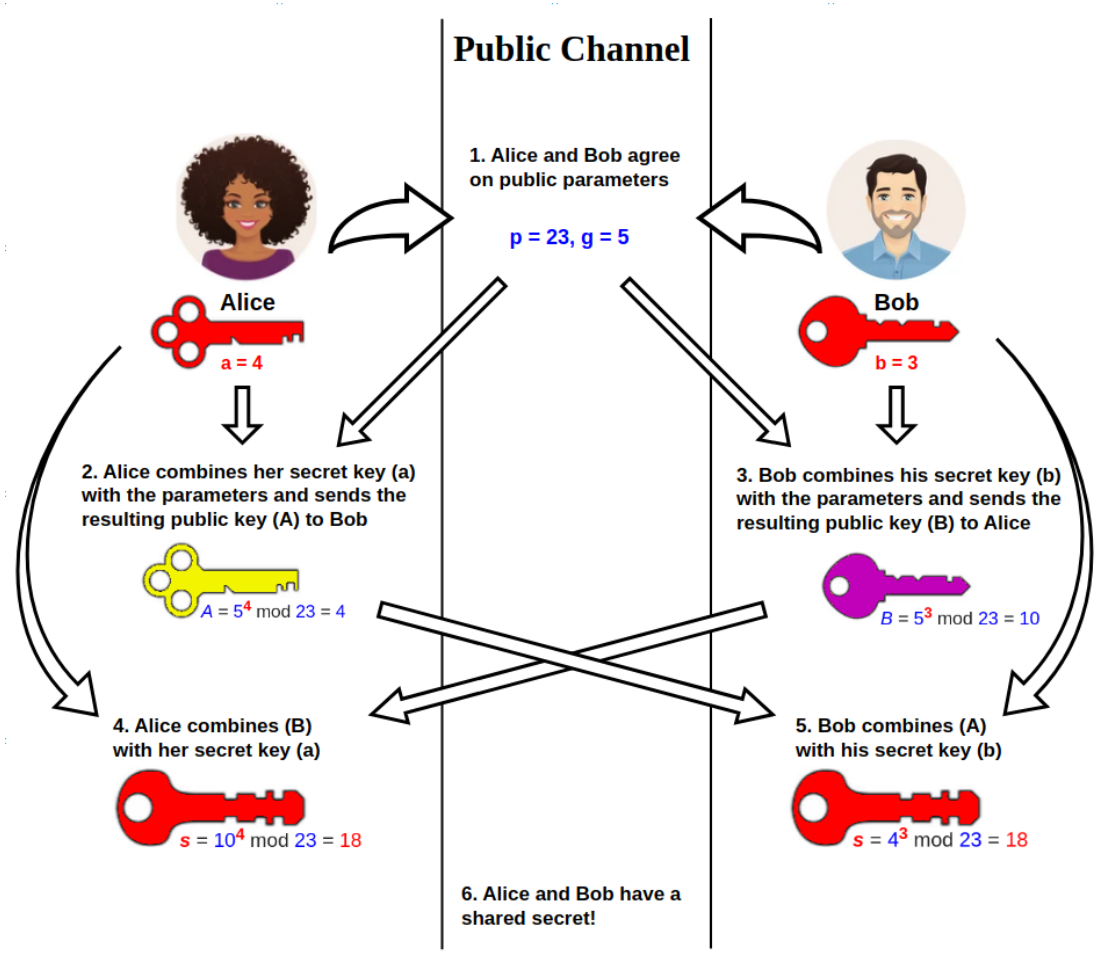


Section 4.2.1:

Diffie-Hellman Key Exchange Mechanism

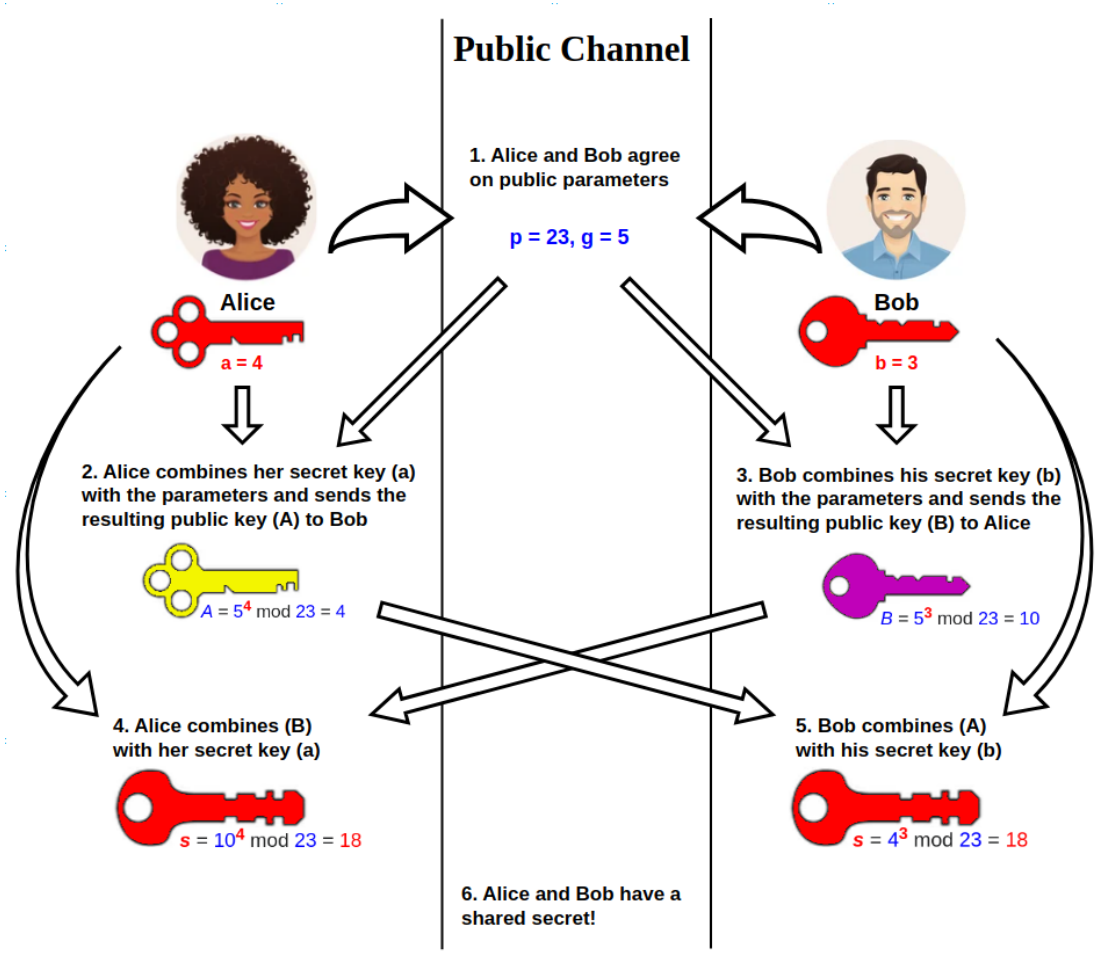
Diffie-Hellman Key Exchange Mechanism

- This mechanism was developed by two cryptographers, Whitefield **Diffie** and Martin **Hellman**, back in 1976.
- Diffile-Hellman (DH) key exchange mechanism is an algorithmic method to:
  - **Exchange an encryption key between two communicating parties via an insecure channel.**
  - **Exchange an encryption key without actually sending it.**
- This mechanism is also called as a key agreement algorithm.



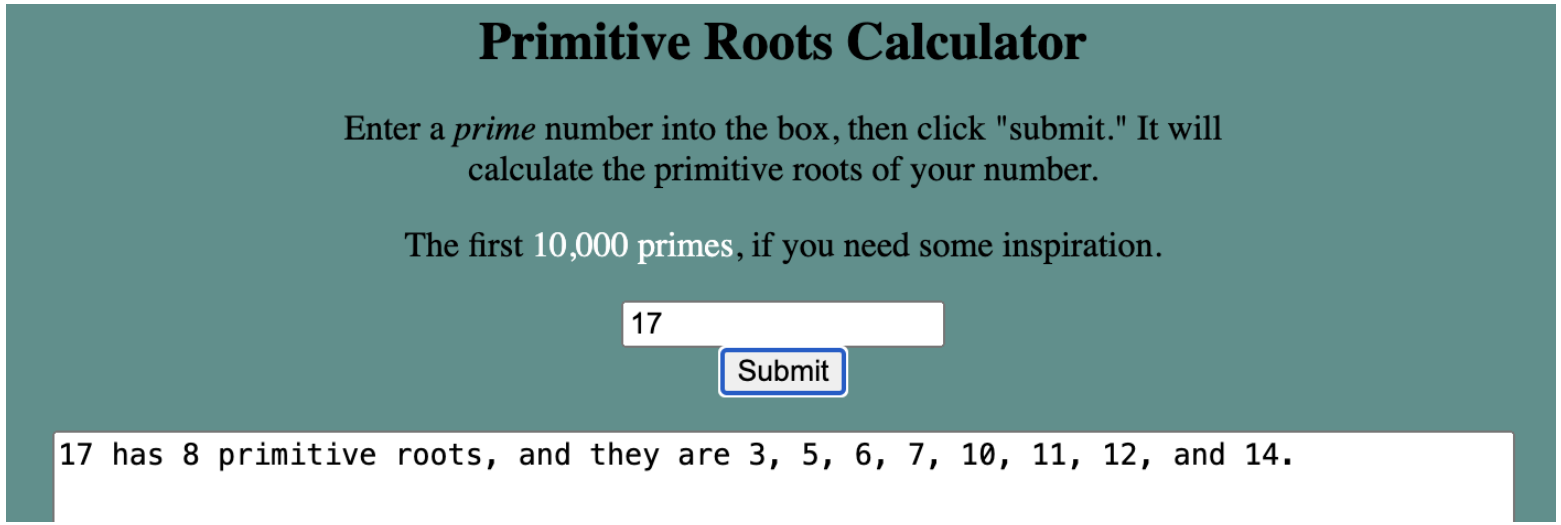
Diffie-Hellman Key Exchange Mechanism

- This mechanism involves **four essential steps**:
  1. **Agree on initial parameters**: Select a shared pair of public values, p (a large prime number) and α (a primitive root modulo p).
  2. **Generate private and public keys**: Each party generates a private key (SK) and then computes a corresponding public key (PK).
  3. **Exchange public keys**: Both parties exchange their public keys over a public channel.
  4. **Compute the shared encryption key (K):** Each party uses the received public key (PK) and their own private key (SK) to compute a shared encryption key, K.



Diffie-Hellman Key Exchange Mechanism

- **Step 1: Agree on initial parameters**: Select a shared pair of public values, p (a large prime number) and α (a primitive root modulo p).
  - Before the key exchange, both communicating parties must agree on the **public parameters**, specifically the values of p and α.
  - p must be **a prime number**, and α must be **a primitive root modulo p**.
  - **For example:**
    - $p = 7, \alpha = 3$ (since 3 is a primitive root modulo 7)
    - $p = 17, \alpha = 7$ (since 7 is a primitive root modulo 17)



Primitive Roots Calculator
http://www.bluetulip.org/2014/programs/primitive.html

$p = 17$
$\alpha = 7$
$SK_{Bob} = 5$
$SK_{Alice} = 2$
$PK_{Bob} = 11$
$PK_{Alice} = 15$

Diffie-Hellman Key Exchange Mechanism

- **Step 2: Generate private and public keys**: Each party generates a private key (SK) and then computes a corresponding public key (PK).
  - Based on the agreed initial values (p and α), each party selects a private key (SK), which is an integer in the range

    $1 \leq SK \leq p - 1$

    - For example, if p = 17, **Bob** might choose $SK_{Bob} = 5$ and **Alice** might choose $SK_{Alice} = 2$
  - Each party then calculates their public key (PK) using the formula: $PK = \alpha^{SK} \bmod p$
    - **For example:**
    - Let p = 17, α = 7, and $SK_{Bob} = 5$ for **Bob**. Then $PK_{Bob} = 7^5 \bmod 17 = 16807 \bmod 17 = \mathbf{11}$
    - Let p = 17, α = 7, and $SK_{Alice} = 2$ for **Alice**. Then $PK_{Alice} = 7^2 \bmod 17 = 49 \bmod 17 = \mathbf{15}$
  - At this point, each party has a pair of keys: a private key (SK), which is **kept secret**, and a public key (PK), which can be **shared**.

$p = 17$
$\alpha = 7$
$SK_{Bob} = 5$
$SK_{Alice} = 2$
$PK_{Bob} = 11$
$PK_{Alice} = 15$

Diffie-Hellman Key Exchange Mechanism

- **Step 3: Exchange public keys:** Both parties exchange their public keys over a public channel.
- **Step 4: Compute the shared encryption key (K):** Each party uses the received public key (PK) and their own private key (SK) to compute a shared encryption key, K.
  - Once the public keys have been exchanged, both parties will have enough information to generate a shared encryption key.
  - Each party computes the agreed encryption key using the following formula:

$$K = (PK_{Received})^{SK_{Own}} \, mod \, p$$

  - **For example: Alice** and **Bob** can compute the shared encryption (K) as follows:
    - **Bob** calculates:

$$\boldsymbol{K} = (PK_{Alice})^{SK_{Bob}} \, mod \, 17 = 15^5 \, mod \, 17 = 759375 \, mod \, 17 = \boldsymbol{2}$$

    - **Alice** calculates:

$$\boldsymbol{K} = (PK_{Bob})^{SK_{Alice}} \, mod \, 17 = 11^2 \, mod \, 17 = 121 \, mod \, 17 = \boldsymbol{2}$$

    - Thus, both Bob and Alice independently compute the same shared encryption key: $K = \mathbf{2}$

Security of Diffie-Hellman Key Mechanism

- As you may have observed, the DH key exchange mechanism **leverages the concept of asymmetric cryptography** – specifically, the use of public and private keys – to securely establish a shared encryption key.
- However, **the encryption process itself remains symmetric**, as both parties use the same agreed-upon key for encryption and decryption.

- A critical component of the DH key exchange is the transmission of public keys between the communicating parties.
- **Question:**
  - Do you have any thoughts on potential issues with this mechanism?
  - Are there any security concerns or vulnerabilities related to the exchange of public keys?

Diffle-Hellman Vulnerability of Man-in-the-Middle Attacks

- At first, Alice and Bob want to communicate with each other in a secure way.
- They adopt the DH key exchange mechanism by preparing **the initial information** and their public-private key pairs (in this case, Y_A is Alice's public key and X_A is Alice's private key).

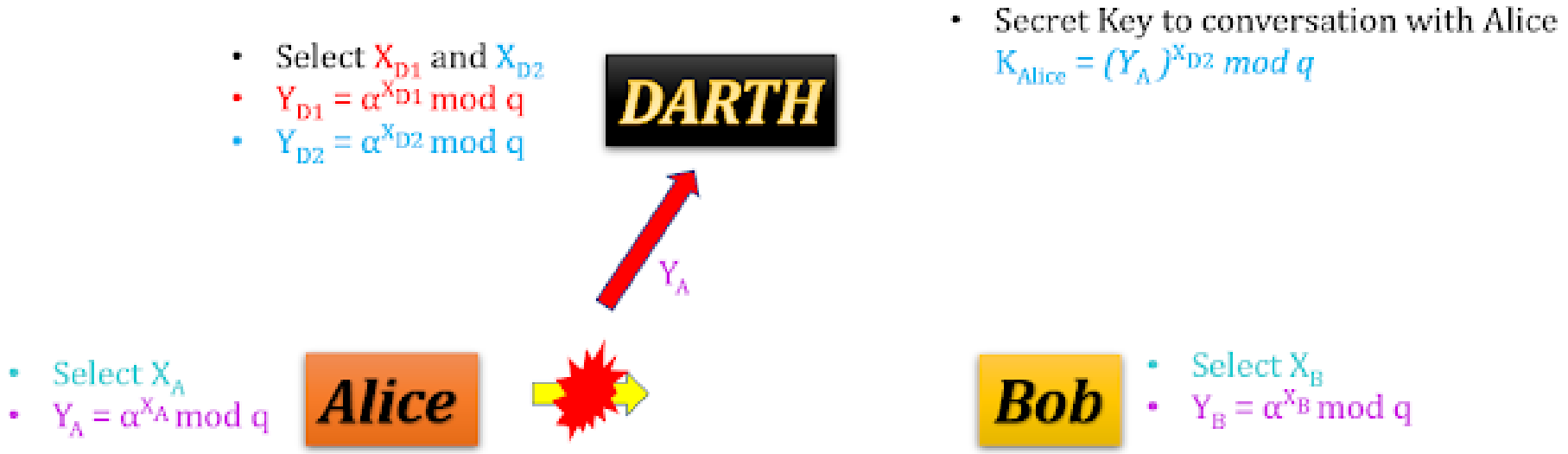
Diffle-Hellman Vulnerability of Man-in-the-Middle Attacks

- Suppose there is **a bad guy (i.e., an adversary)** who wants to capture and eavesdrop messages between Bob and Alice. In this situation, let's call this bad guy ***Darth***.

Diffle-Hellman Vulnerability of Man-in-the-Middle Attacks

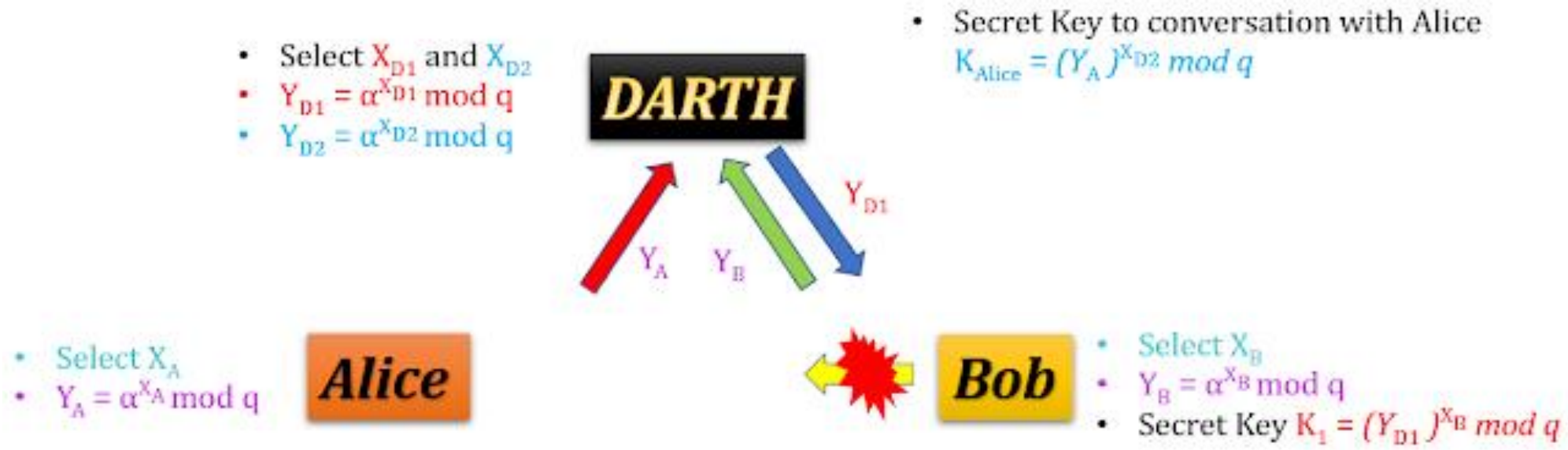
- With the intention to eavesdrop the communication, **Darth** prepares two key pairs.
- In this example, X_{D1} is a private key and Y_{D1} is a public key to attack **Bob**. Also, X_{D2} is a private key and Y_{D2} is a public key to attack **Alice**.

Diffie-Hellman Vulnerability for Man-in-the-Middle Attacks



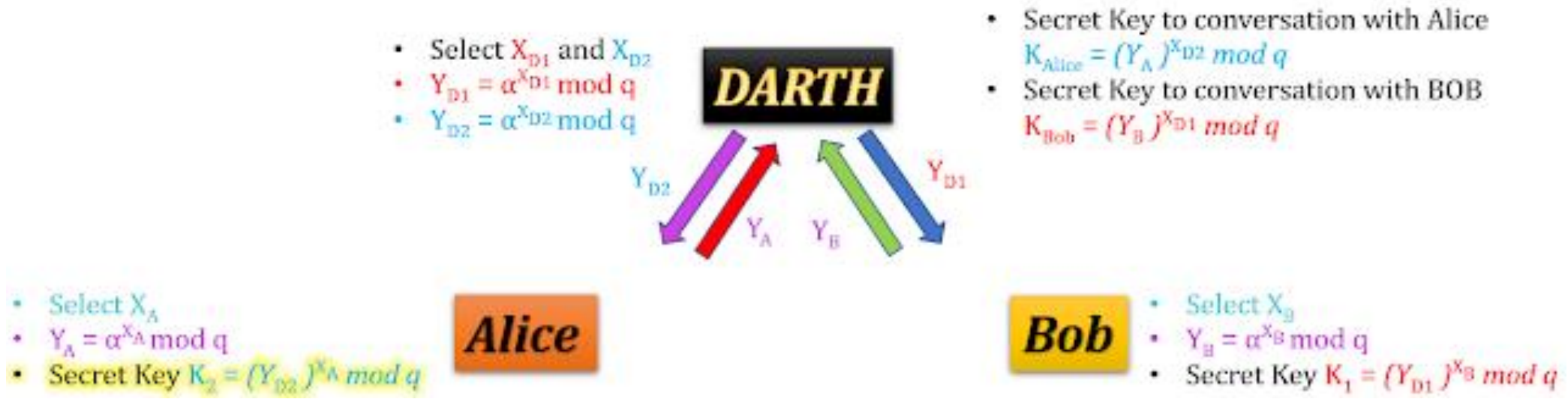
- When Alice exchanges her public key (Y_A) to Bob, Darth captures the communication and receives Alice's public key.
- With Alice's public key, Darth can calculate an encryption key for Alice (K_{Alice})

Diffie-Hellman Vulnerability for Man-in-the-Middle Attacks



- Darth can do the same thing when Bob exchange his public key (Y_B) to Alice.
- Instead of sending the other public keys, Darth exchanges his public key to both victims.
  For instance, Darth sends Y_{D1} to Bob instead of Y_A.

Diffie-Hellman Vulnerability for Man-in-the-Middle Attacks

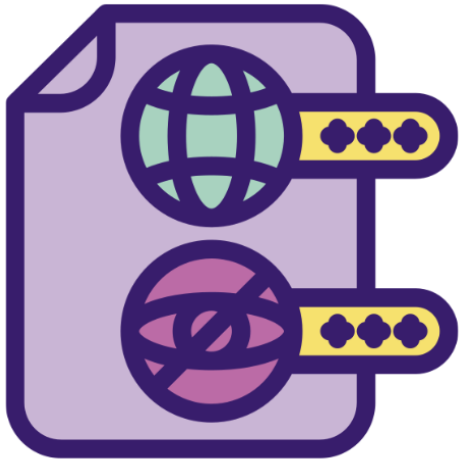


- Now, Alice, Bob, and Darth calculate an agreed encryption key from the received public key.
- In this case, Alice may think that Darth's public key (Y_{D2}) is Bob's public key (Y_B).
- With this Man-in-the-Middle attack, both victims will not know that they were attacked.

Security of Diffie-Hellman Key Exchange Mechanims

- Diffie-Hellman key exchange algorithm uses **mathematical properties** to securely exchange public key between two communicating parties.
- Diffie-Hellman key exchange algorithm uses **public keys to calculate an agreed key** for encryption and decryption.
  - The encryption and decryption process can be done as identical as symmetric key cryptography.
- Diffie-Hellman key exchange algorithm is **vulnerable to the Man-in-the-Middle attack**, which we see from the situation of Darth attacking Bob and Alice.
- It can be concluded that Diffie-Hellman key exchange algorithm is not secure anymore **without the secure communication channel**.

***Question:* How can we protect against the Man-in-the-Middle attack?**

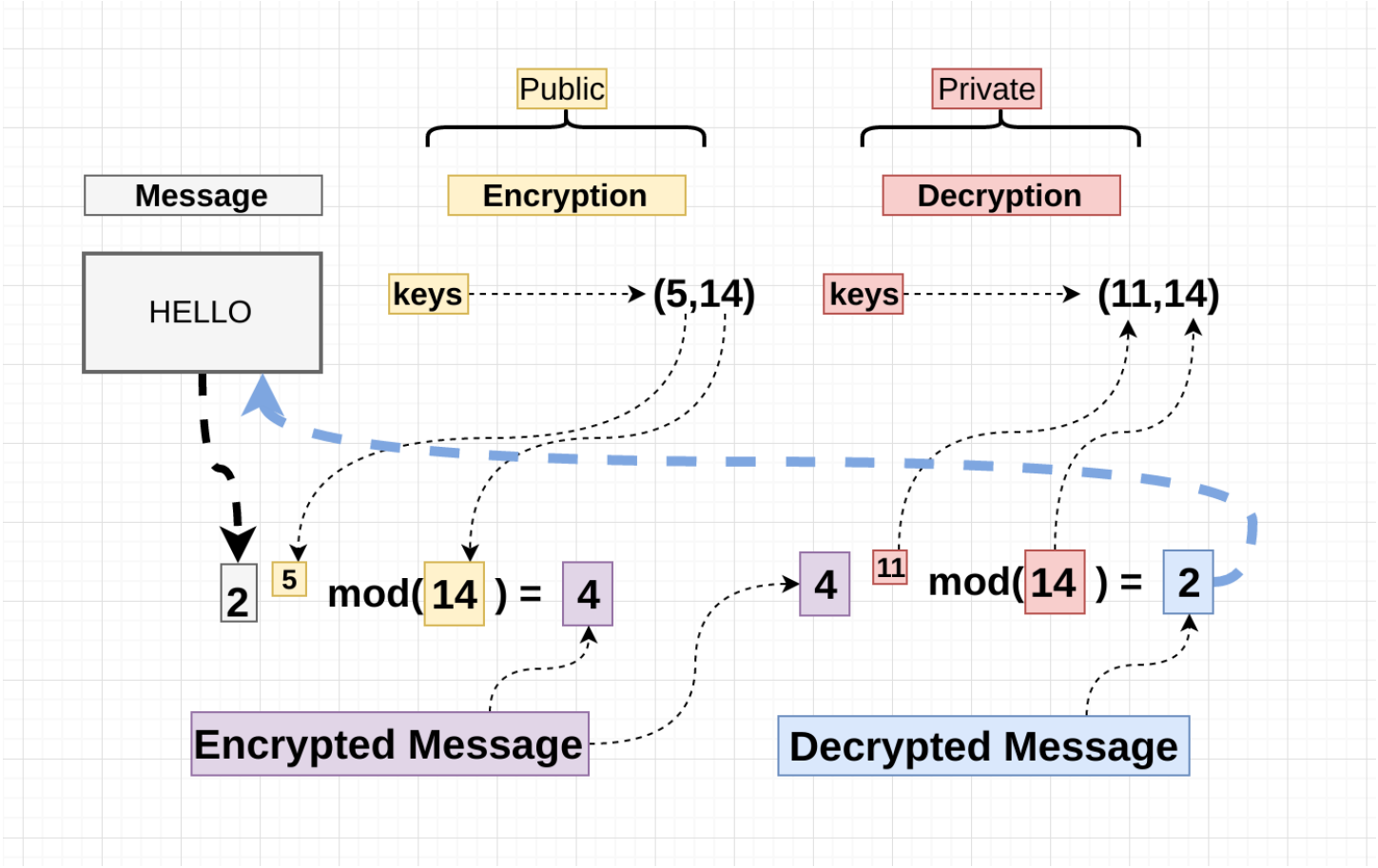


Section 4.2.2:

RSA Algorithm

RSA Algorithm

- Another encryption algorithm for asymmetric key cryptography is **RSA algorithm**.
- It was developed by Ron **R**ivest, Adi **S**hamir, and Len **A**delman (**RSA**) as a block cipher using different keys



RSA Algorithm

- The RSA algorithm generates **a public–private key pair** through **six steps**:
  1. Select two distinct prime numbers p and q such that $p \neq q$.
  2. Compute their product: $n = p \times q$.
  3. Calculate Euler's totient function: $\Phi(n) = (p-1)(q-1)$.
  4. Choose **a public key** e such that e is relatively prime to $\Phi(n)$.
  5. Compute **a private key** d as the modular multiplicative inverse of e with respect to $\Phi(n)$.
  6. Form the public and private key pair: the public key is (e, n) ,and the private key is (d, n).

Standard RSA Algorithm

Phase 1: Key Generation

1: Initialize two random large primes as p, q
2: Evaluate modulus n as,
$n = p*q$
3: Find Euler totient function as,
$\varphi(n) = (p-1)*(q-1)$
4: Now, compute public key exponent e, such that,
$1 < e < \varphi(n)$, and GCD of $(e, \varphi(n)) = 1$
5: And compute private key exponent d, such that,
$e*d = 1 \bmod \varphi(n)$
6: Generated Public key is (n, e), and Private key is (n, d)

Phase 2: Encryption

$c = m^e \bmod n$,
that uses the public key components as (n, e),
and Plaint text message as m

Phase 3: Decryption

$m = c^d \bmod n$,
that uses the private key components as (n, d),
where c is the encoded cipher string and m is the original string.

RSA Algorithm

1. Select two distinct prime numbers p and q such that $p \neq q$.

$$p = 11 \qquad q = 13$$

2. Compute their product: $n = p \times q$.

$$n = p \times q = 11 \times 13 = 143$$

3. Calculate Euler's totient function: $\Phi(n) = (p-1)(q-1)$.

$$\Phi(n) = (p-1)(q-1) = 10 \times 12 = 120$$

Standard RSA Algorithm

Phase 1: Key Generation

1: Initialize two random large primes as p, q
2: Evaluate modulus n as,
n = p*q
3: Find Euler totient function as,
φ(n) = (p-1)*(q-1)
4: Now, compute public key exponent e, such that,
1 < e < φ(n), and GCD of (e, φ(n)) = 1
5: And compute private key exponent d, such that,
e*d = 1*mod φ(n)
6: Generated Public key is (n, e), and Private key is (n, d)

Phase 2: Encryption

$c = m^e \bmod n$,
that uses the public key components as (n, e),
and Plaint text message as m

Phase 3: Decryption

$m = c^d \bmod n$,
that uses the private key components as (n, d),
where c is the encoded cipher string and m is the original string.

RSA Algorithm

4. Choose **a public key e** such that e is relatively prime to $\Phi(n)$.
   - From the definition, p and q are relatively prime or coprime to each other if $GCD(p, q) = 1$.
   - In this step, we will choose a public key e that is relatively prime to $\Phi(n)$ such that $GCD(\Phi(n), e) = 1$.

$$\boldsymbol{GCD(\Phi(n), e) = GCD(120, 13) = 1}$$

Standard RSA Algorithm

Phase 1: Key Generation

1: Initialize two random large primes as p, q
2: Evaluate modulus n as,
n = p*q
3: Find Euler totient function as,
φ(n) = (p-1)*(q-1)
4: Now, compute public key exponent e, such that,
1 < e < φ(n), and GCD of (e, φ(n)) = 1
5: And compute private key exponent d, such that,
e*d = 1*mod φ(n)
6: Generated Public key is (n, e), and Private key is (n, d)

Phase 2: Encryption

$c = m^e \bmod n$,
that uses the public key components as (n, e),
and Plaint text message as m

Phase 3: Decryption

$m = c^d \bmod n$,
that uses the private key components as (n, d),
where c is the encoded cipher string and m is the original string.

RSA Algorithm

4. Compute **a private key d** as the modular multiplicative inverse of e with respect to $\Phi(n)$.
   - The RSA algorithm defines the private key using the formula:

$$d = e^{-1} \bmod \Phi(n).$$

   - Since the private key $d$must be an integer, the formula can be rewritten as:

$$d = \frac{(\Phi(n) \cdot i) + 1}{e}, for\ i = 1,2,3, \ldots \in \mathbb{N}^{+},$$

   - until d evaluates to an integer.

For $i = 1$; $d = \frac{(120 \cdot 1)+1}{13} = 9.31$

For $i = 2$; $d = \frac{(120 \cdot 2)+1}{13} = 18.54$

For $i = 3$; $d = \frac{(120 \cdot 3)+1}{13} = 30.08$

For $i = 4$; $d = \frac{(120 \cdot 4)+1}{13} = 37$

$\boldsymbol{d = 37}$

Standard RSA Algorithm

Phase 1: Key Generation

1: Initialize two random large primes as p, q
2: Evaluate modulus n as,
n = p*q
3: Find Euler totient function as,
φ(n) = (p-1)*(q-1)
4: Now, compute public key exponent e, such that,
1 < e < φ(n), and GCD of (e, φ(n)) = 1
5: And compute private key exponent d, such that,
e*d = 1*mod φ(n)
6: Generated Public key is (n, e), and Private key is (n, d)

Phase 2: Encryption

$c = m^e \bmod n$,
that uses the public key components as (n, e),
d Plaint text message as m

Phase 3: Decryption

$m = c^d \bmod n$,
that uses the private key components as (n, d),
where c is the encoded cipher string and m is the original string.

RSA Algorithm

5. Form the public and private key pair: the public key is (e, n), and the private key is (d, n).

$$PK = (e, n) = (13, 143)$$

$$SK = (d, n) = (37, 143)$$

Standard RSA Algorithm

Phase 1: Key Generation

1: Initialize two random large primes as p, q
2: Evaluate modulus n as,
n = p*q
3: Find Euler totient function as,
φ(n) = (p-1)*(q-1)
4: Now, compute public key exponent e, such that,
1 < e < φ(n), and GCD of (e, φ(n)) = 1
5: And compute private key exponent d, such that,
e*d = 1*mod φ(n)
6: Generated Public key is (n, e), and Private key is (n, d)

Phase 2: Encryption

$c = m^e \bmod n$,
that uses the public key components as (n, e),
and Plaint text message as m

Phase 3: Decryption

$m = c^d \bmod n$,
that uses the private key components as (n, d),
where c is the encoded cipher string and m is the original string.

RSA Algorithm

- Now, we obtain a public-private key pair

$$PK = (e, n) = (13, 143)$$

$$SK = (d, n) = (37, 143)$$

- RSA algorithm needs to encode each character in **an integer**.
- RSA algorithm defines the encryption function as:

$$c = m^e mod\ n$$

$$c = 18^{13} mod\ 143 = 57$$

- RSA algorithm defines the decryption function as:

$$m = c^d mod\ n$$

$$m = 57^{37} mod\ 143 = 18$$

Big number calculator: https://www.calculator.net/big-number-calculator.html

Standard RSA Algorithm

Phase 1: Key Generation

1: Initialize two random large primes as p, q
2: Evaluate modulus n as,
n = p*q
3: Find Euler totient function as,
φ(n) = (p-1)*(q-1)
4: Now, compute public key exponent e, such that,
1 < e < φ(n), and GCD of (e, φ(n)) = 1
5: And compute private key exponent d, such that,
e*d = 1*mod φ(n)
6: Generated Public key is (n, e), and Private key is (n, d)

Phase 2: Encryption

$c = m^e \bmod n$,
that uses the public key components as (n, e),
and Plaint text message as m

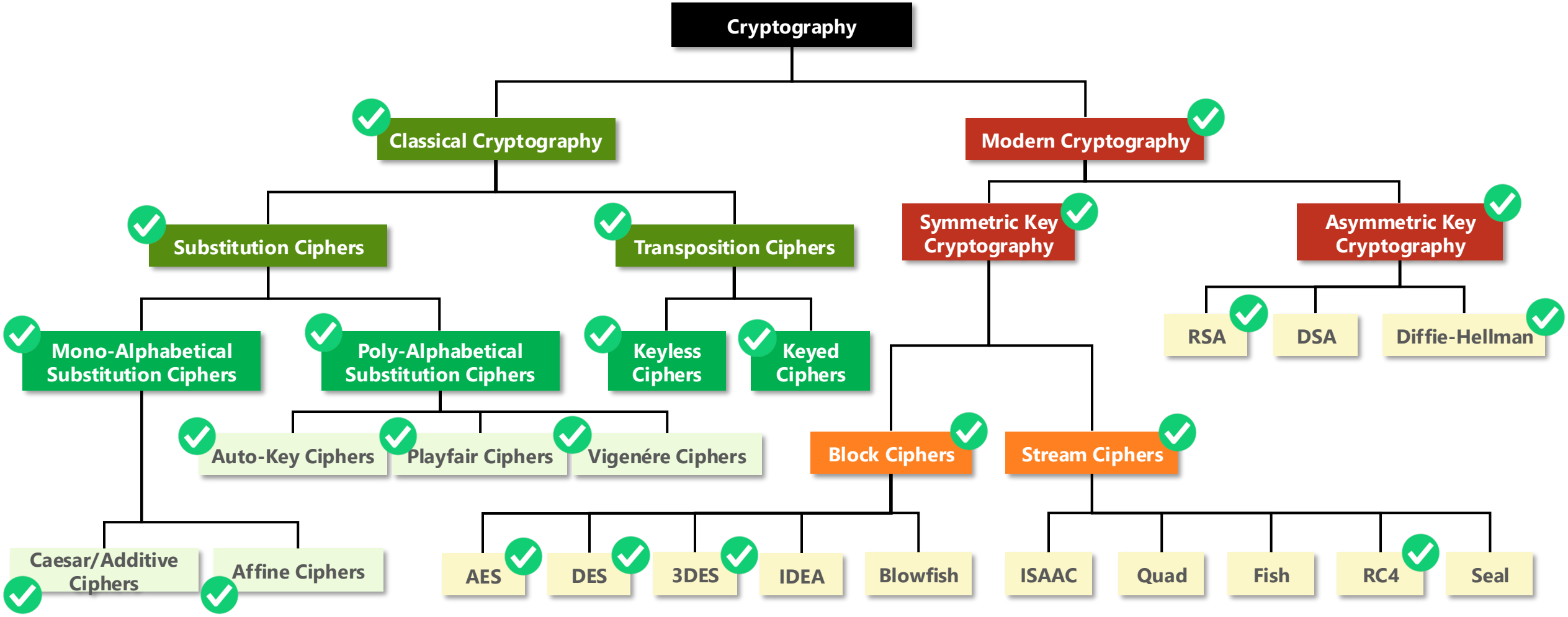
Phase 3: Decryption

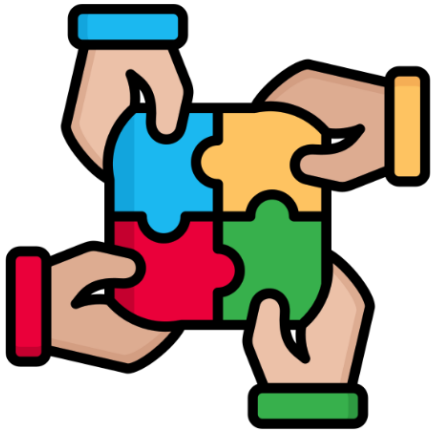
$m = c^d \bmod n$,
that uses the private key components as (n, d),
where c is the encoded cipher string and m is the original string.

A Summary of RSA Algorithm

- The RSA algorithm is defined as **an asymmetric key block cipher**.
- It employs the concept of **relative primality** to generate a pair of public and private keys.
- It specifies how to encrypt a plaintext message and decrypt a ciphertext message, both represented **in decimal or integer format**.
- The RSA algorithm requires ***significant computational power*** to perform encryption and decryption due to its underlying **mathematical complexity**.

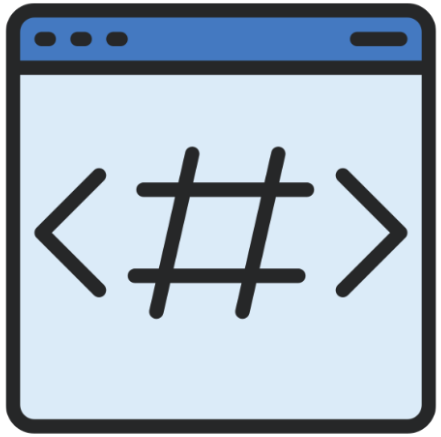
Classification of Cryptographic Techniques





Section 5:

Data Integrity Assurance



Section 5.1:

Hash Function

Are Keyless Cryptographic Techniques Still Working?

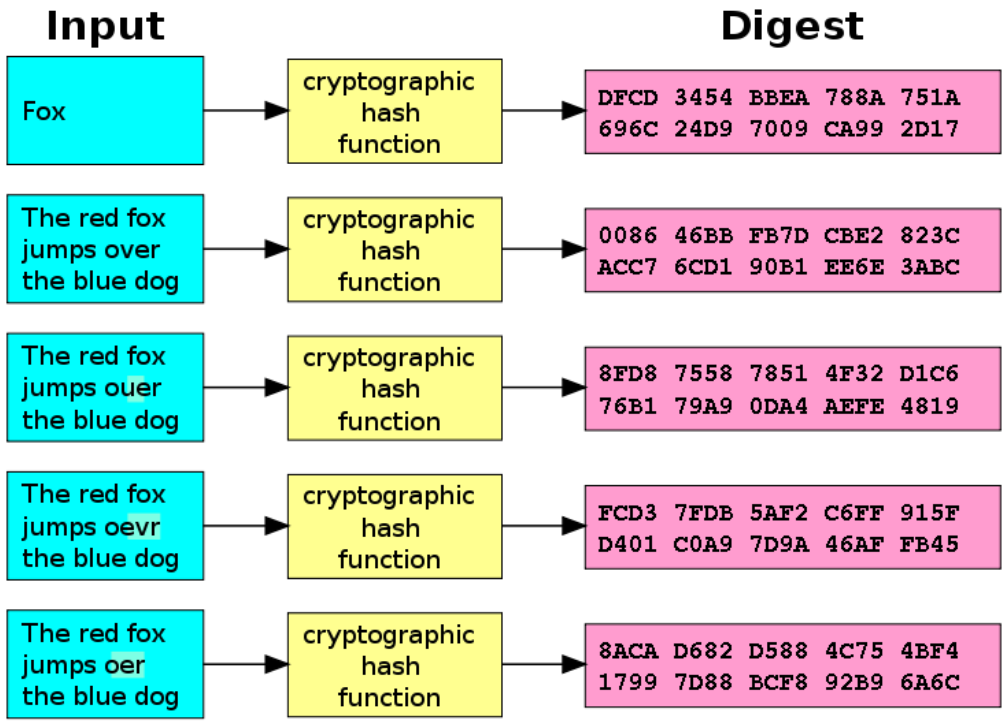
- We used to discuss that keyless cryptography is weak when the algorithm is simple and exposed to anyone.
- **Question:** Are we still able to use keyless cryptography in modern information system?

- **Answer: Yes!**

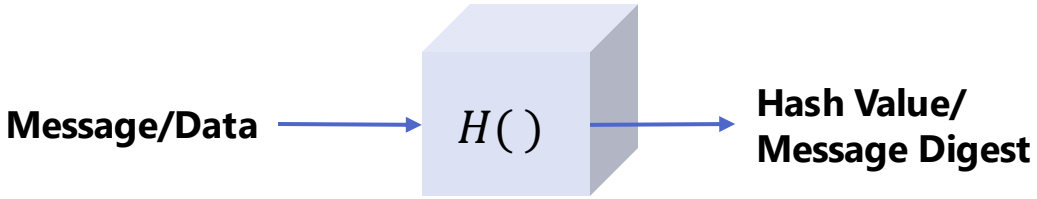
- But we use no-key cryptography for other purposes than the protection of confidentiality.
- Most of the time, modern cryptography uses **keyless algorithm to ensure other properties except the confidentiality**, like **data integrity**, along with keyed cryptography (such as symmetric key and asymmetric key cryptography).

Hash Function

- **Hash function** is a type of modern keyless cryptographic techniques that uses for ensuring data integrity.



- **A cryptographic hash function** H is **a one-way function** that maps **an input of arbitrary length** (or blocks of input) to **a fixed-size output**.
  - **A one-way function** means that *it is computationally infeasible to invert* the output to recover the original input.
  - The output of a hash function is typically referred to as the ***hash value*** or ***message digest***.



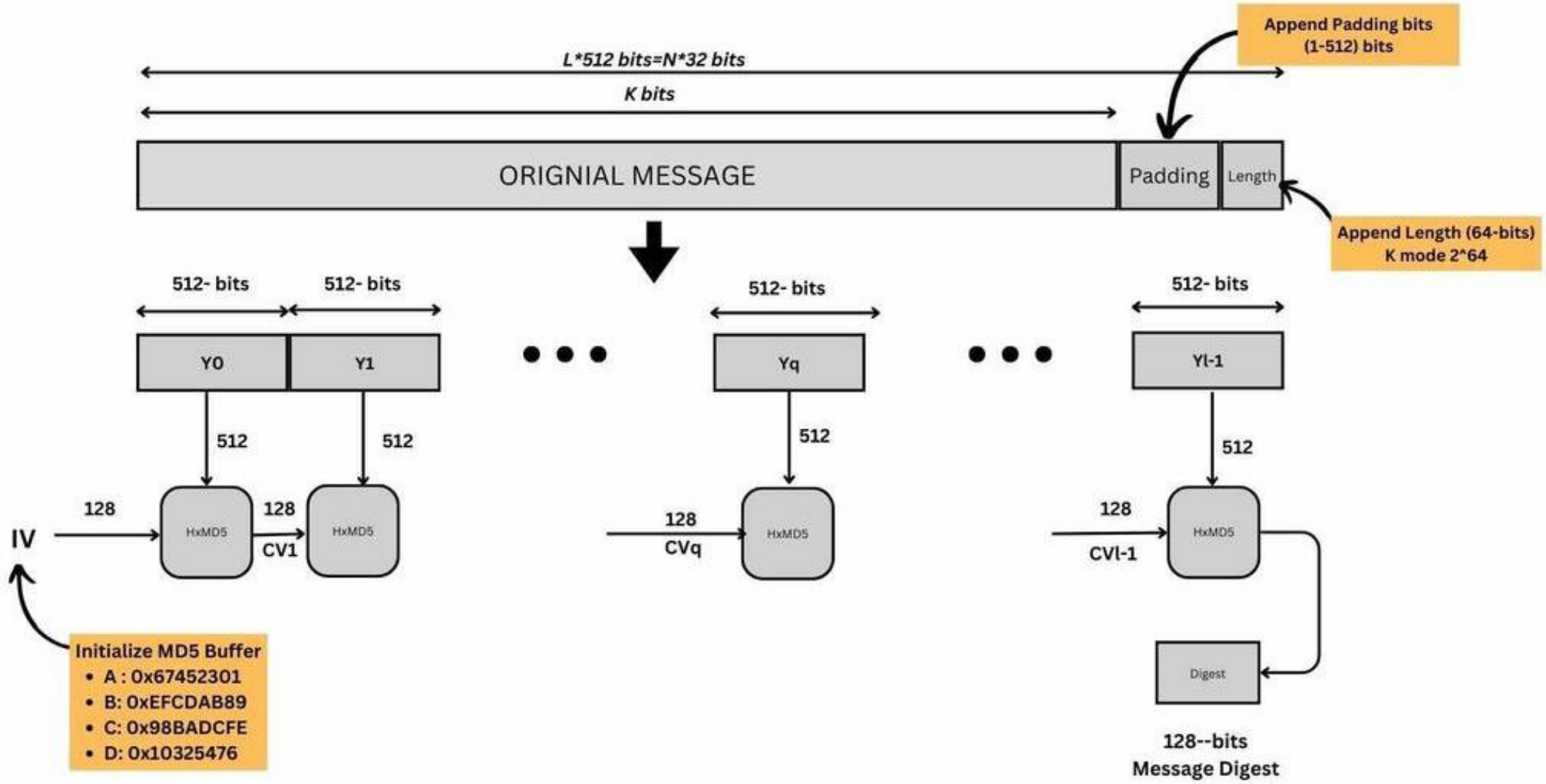
Properties of Hash Function

- **Compression:** The output of a hash function is typically much smaller than its input. For example, the message digest of a 100 MB file can be represented as a 256-bit string.
- **Pre-image resistance:** It is computationally difficult to determine the original input from a given hash value.
  - That is, given $h = H(M)$ as the hash value of a message M, it is infeasible to recover $M$from h.
- **Weak collision resistance:** Given a message M_1 ,it is computationally hard to find another message M_2 such that $H(M_1) = H(M_2)$.
  - In other words, it is infeasible to find two distinct messages that produce the same hash value.
- **Strong collision resistance:** The probability that two different messages $M_1$and M_2 yield the same hash value $H(M_1) = H(M_2)$ is negligible.

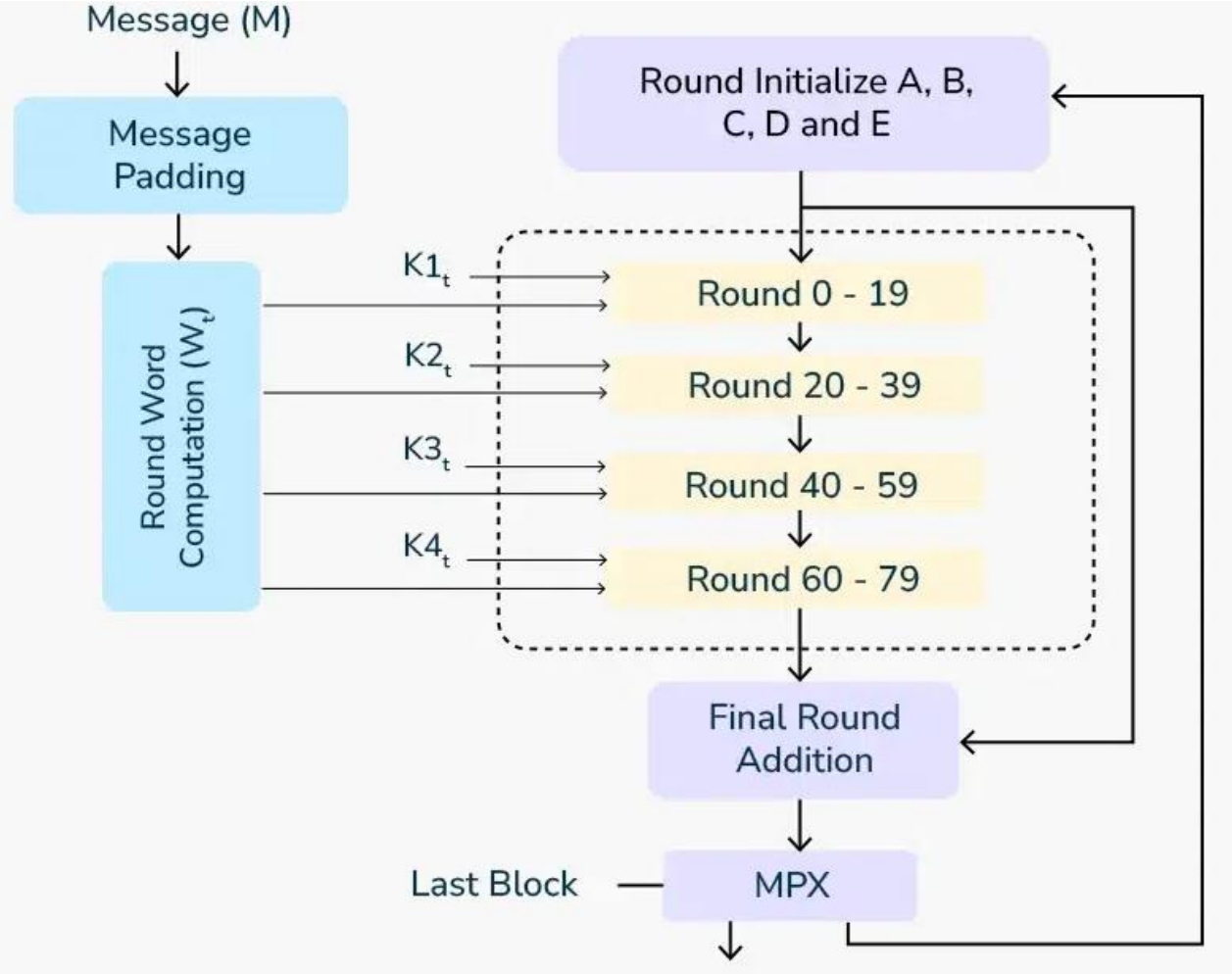
Characteristics of Good Hash Function

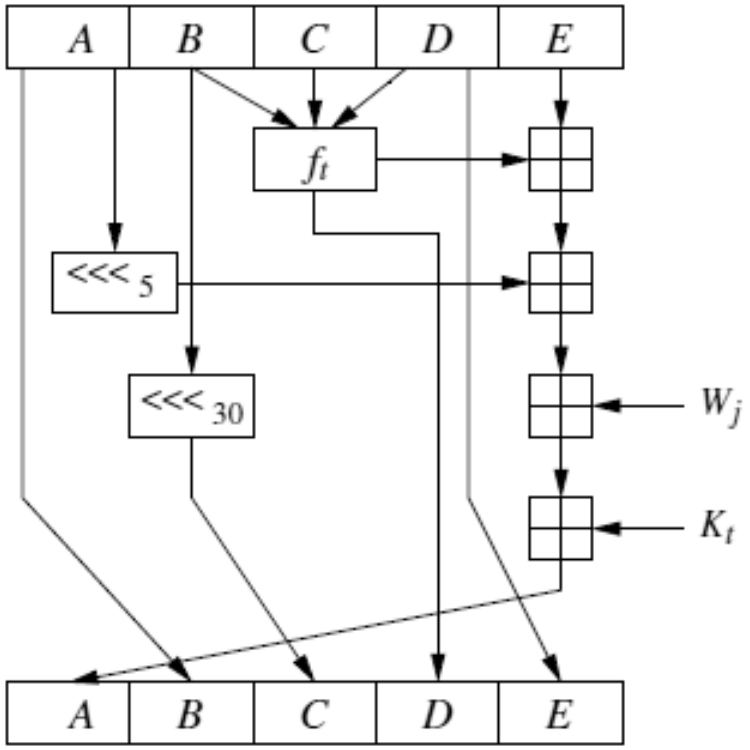
- **Efficiency:** It is quick and efficient to compute the hash value for any given input.
- **Input Flexibility:** The function can process data blocks of variable length.
- **Avalanche Effect:** A small change in the input message should result in a significantly different hash value.
- **One-Wayness:** It is computationally infeasible to reverse-engineer the original message from its hash value.
- **Full Utilization of Input:** The hash function should incorporate all input data into its computation.
- **Uniform Distribution:** It should uniformly distribute hash values across the output space to avoid clustering.
- **Collision Resistance:** It should generate distinct hash values even for similar messages, minimizing the chance of collisions.

Message Digest 5 (MD5)



Secure Hash Algorithm (SHA)





Sr. No.	Parameters	SHA-1	SHA-256	SHA-384	SHA-512
1	Message digest size	160	256	384	512
2	Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
3	Block size	512	512	1024	1024
4	Word size	32	32	64	64
5	Number of steps	80	64	80	80
6	Security level	80	128	192	256

Note: All values are measured in bits.

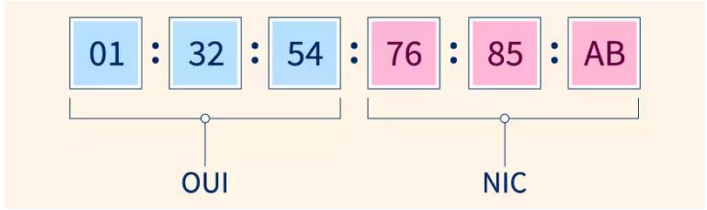
SHA-1 Versus MD5

Secure Hash Algorithm 1 (SHA-1)	Message Digest 5 (MD5)
It produces 160-bit message digest size.	It produces 128-bit message digest size.
It is strong against brute-force attacks.	It is weak against brute-force attacks.
It is resistant to differential cryptanalysis.	It is vulnerable to differential cryptanalysis.
It is slower than MD5 to produce message digests.	It is faster than SHA-1 to product message digests.
It uses big-endian method.	It use little-endian method.
It operates 4 steps in 20 iterations = 80 steps.	It operate 4 steps in 16 rounds = 64 steps.

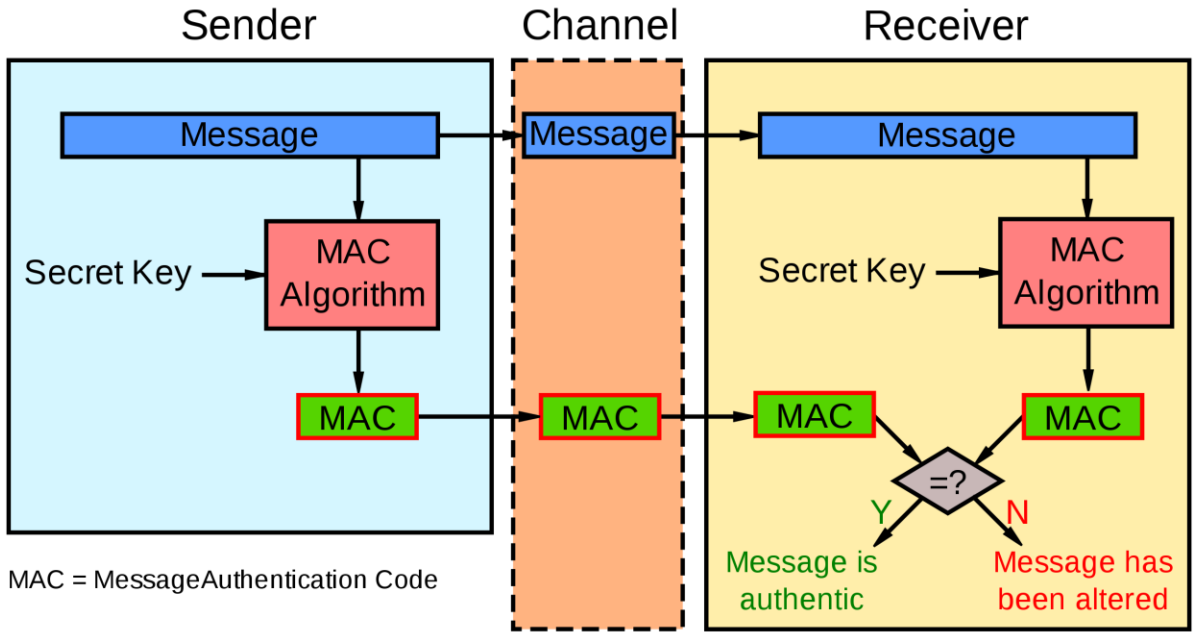
Section 5.2:

Message Authentication Code

An Introduction to Message Authentication Code (MAC)



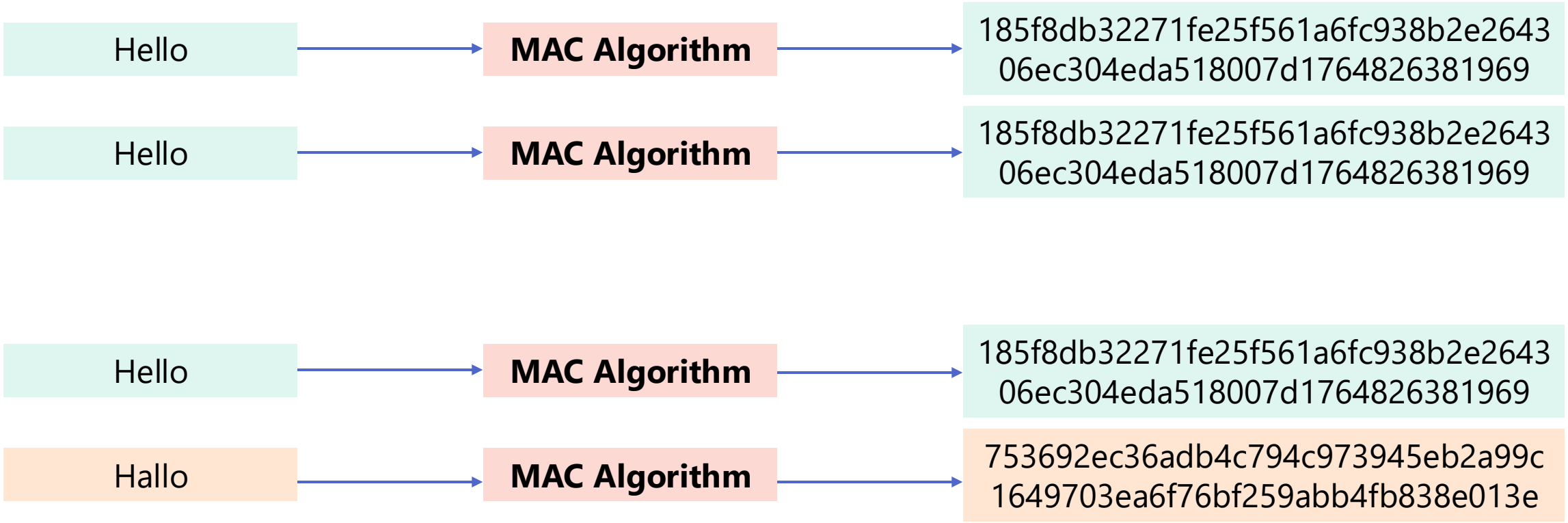
- **Message Authentication Code or MAC** is another application of cryptography used to ensure message authenticity, verifying that the received message genuinely originates from the sender and has not been altered.



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

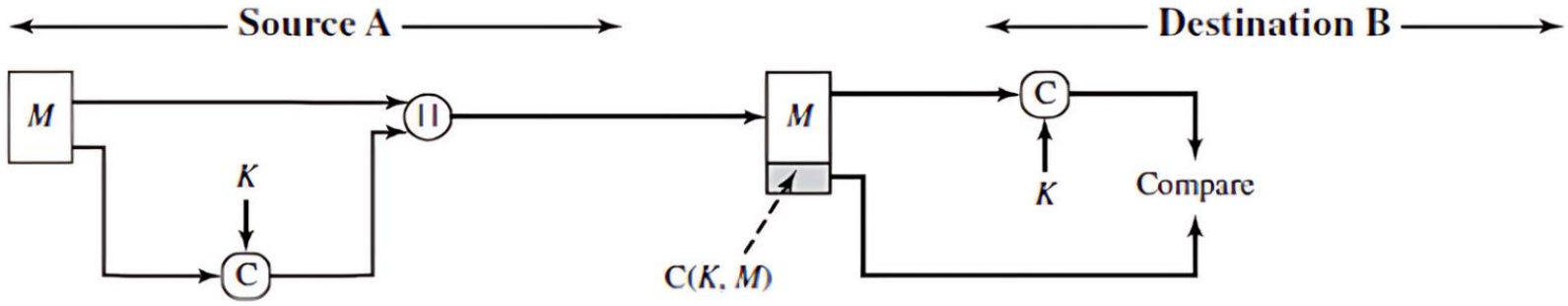
$$A = MAC(K, M) = H(K||M)$$

An Example of MAC Use Case



Use Cases of Message Authentication Code

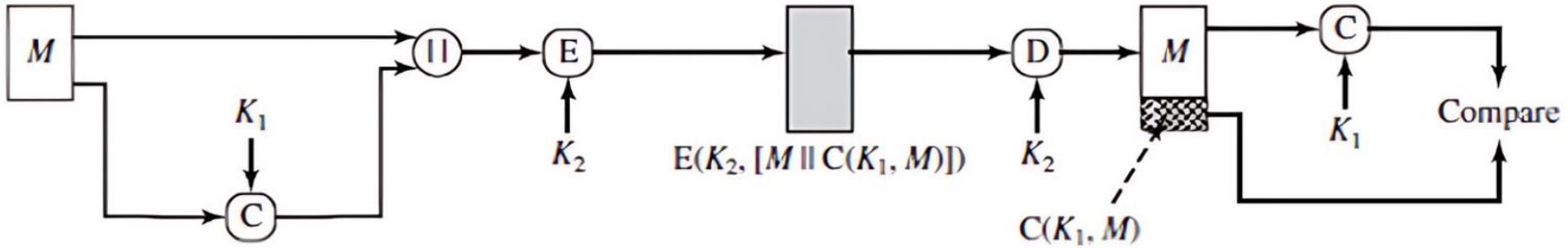
- **Case 1: Message Authentication Only**



- **A** wants to send an authentic message to **B**. The first requirement is that **A** and **B** share **a secret key (K)**.
- **A** calculates a MAC value (A_1) from the message M and the key K using ***a MAC function (C)***: $A_1 = C(M, K)$
- **A** appends the MAC value to the message, forming: $A_1||M$ and sends this to **B**.
- Upon receiving the message, **B** uses the shared key (K) to calculate their own MAC value (A_2) over the received message M: $A_2 = C(M, K)$
- **B** compares A_1 and A_2. If $A_1 = A_2$, the message M is authenticated. If not, the message may have been altered or forged.

Use Cases of Message Authentication Code

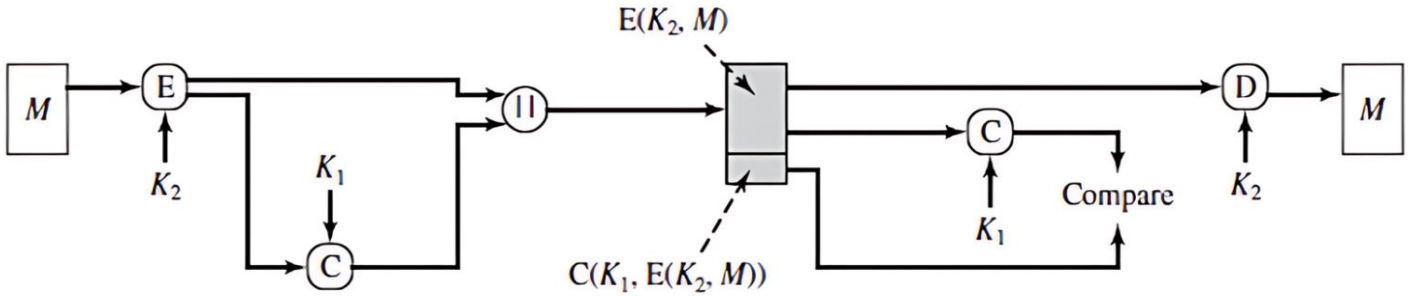
- **Case 2: Message Authentication and Confidentiality – Authentication Tied to Plaintext:**



- **A** wants to send a message M to **B** with both authentication and confidentiality. To do this, **A** and **B** must share two secret keys: K_1 for authentication (MAC), and K_2 for encryption
- **A** calculates a MAC value (A_1) over the plaintext message (M) using key K_1 and **a MAC function (C)**: $A_1 = C(M, K_1)$
- **A** appends the MAC value to the plaintext message, forming: $A_1 || M$ and then encrypts the entire package using key K_2 to produce the ciphertext X. Then, **A** sends X to **B**.
- Upon receiving X, **B** uses K_2 to decrypt the ciphertext and retrieve: $A_1 || M$
- **B** then uses K_1 to compute their own MAC value (A_2) from the decrypted message M: $A_2 = C(M, K_1)$
- **B** compares A_1 and A_2: If $A_1 = A_2$, the message is confirmed to be authentic and untampered. Otherwise, it is considered compromised.

Use Cases of Message Authentication Code

- **Case 3: Message Authentication and Confidentiality – Authentication Tied to Ciphertext:**



- **A** wants to send a message M to **B** with both confidentiality and authentication. **A** and **B** must share two secret keys: K_1 for authentication (MAC), and K_2 for encryption
- **A** encrypts the plaintext message M using K_2, producing a ciphertext X.
- **A** computes a MAC value (A_1) over the ciphertext X using K_1 and **a MAC function (C)**: $A_1 = C(X, K_1)$
- **A** sends the combined message to **B** in the format: $A_1 || X$
- Upon receiving the message, **B** first uses K_1 to compute their own MAC value (A_2) over the received ciphertext X: $A_2 = C(X, K_1)$
- **B** then compares A_1 and A_2: If $A_1 = A_2$, the ciphertext is authenticated and untampered. If not, the message is rejected.
- If the MAC is valid, **B** proceeds to decrypt the ciphertext X using K_2 to retrieve the original message M.

Security for Using Hash Function for Message Authentication Code

- We discussed earlier that:

$$A = C(M, K) = H(M||K)$$

- The question is "**Is it secure to use the hash function (H) like MD5 and SHA-1 as the MAC function for message authentication?**"

- **Answer: NO!** Both **MD5** and **SHA-1** have inherently **weak designs**, and **advancements in cryptanalysis** have exposed them to serious vulnerabilities.
- The most critical issue with these algorithms is the possibility of **a collision**—a situation where two different inputs produce the same hash value. Collisions undermine the integrity, and authenticity guarantees of cryptographic systems, making MD5 and SHA-1 unsuitable for secure applications such as digital signatures, certificate validation, and data integrity checks.

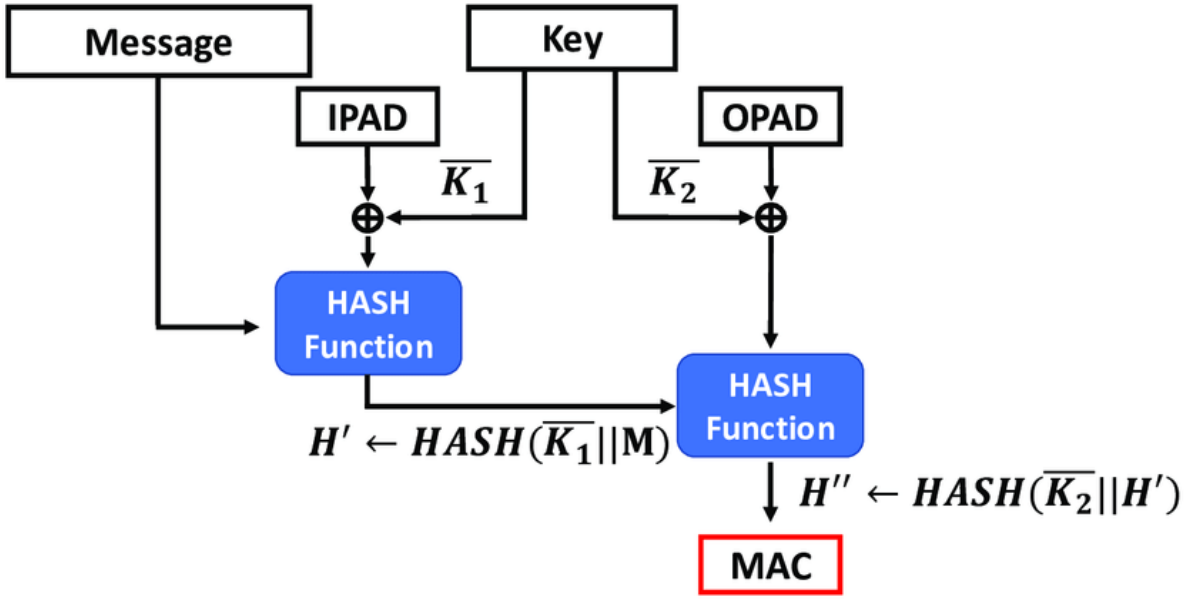
Hash-based Message Authentication Code (HMAC)

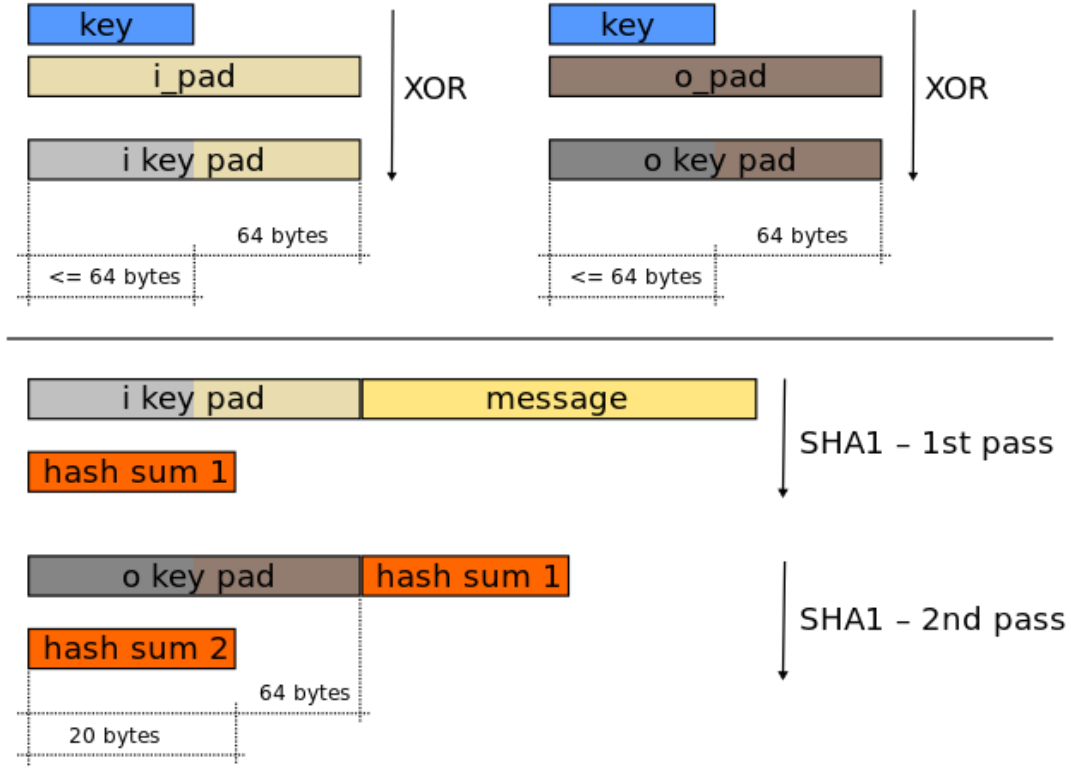
- A solution to overcome the weakness of using hash functions as the MAC function is to use **a Key-Hash MAC or Hash-based MAC (HMAC)**.

- **HMAC (Hash-based Message Authentication Code)** is a cryptographic technique used to ensure message authentication and data integrity. It combines a cryptographic hash function (such as SHA-256) with a secret key to produce a unique MAC.
- HMAC is based on the principles of symmetric key cryptography, meaning that both the sender and receiver must share the same secret key in advance.
- It is **widely used** in secure protocols like TLS, IPSec, and HTTPS, as it offers strong resistance to cryptographic attacks, including length extension and collision attacks.
- HMAC introduces a cryptographic construction that **mitigates the reliance on the collision resistance** of the underlying hash function. This design makes HMAC more robust and secure compared to traditional MAC methods, even when the hash function itself is partially vulnerable.

Hash-based Message Authentication Code (MAC)

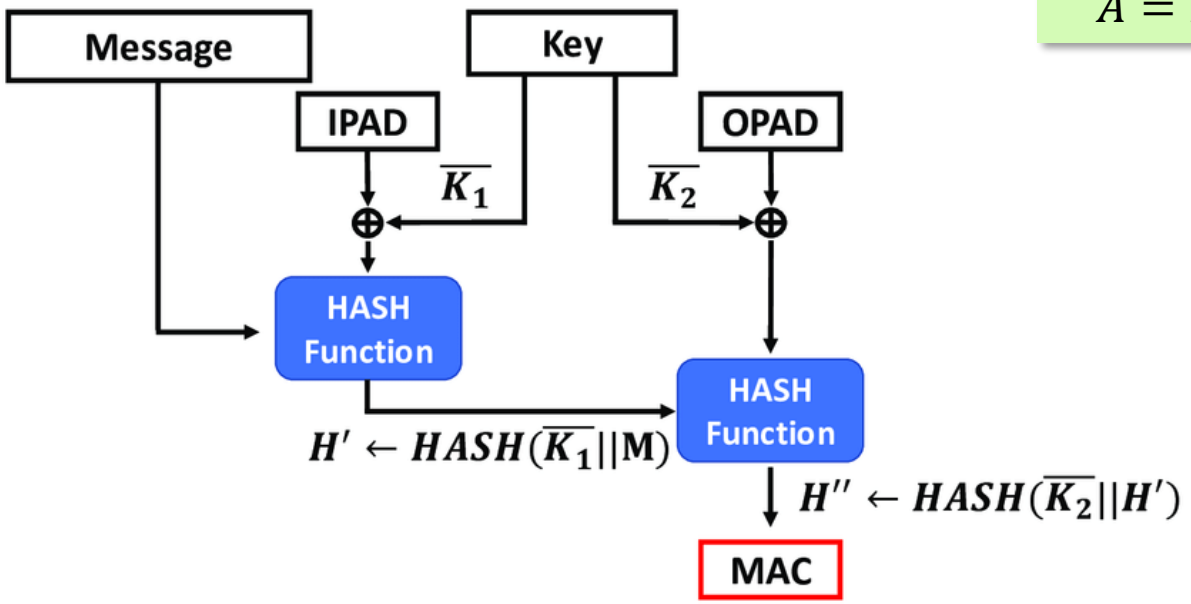
- **HMAC (Hash-based Message Authentication Code)** is a cryptographic technique used to ensure message authentication and data integrity. It combines a cryptographic hash function (such as SHA-256) with a secret key to produce a unique MAC.





Hash-based Message Authentication Code (MAC)

- **HMAC (Hash-based Message Authentication Code)** is a cryptographic technique used to ensure message authentication and data integrity. It combines a cryptographic hash function (such as SHA-256) with a secret key to produce a unique MAC.



$$A = HMAC(K, M) = H((K' \oplus opad)||H((K' \oplus ipad)||M))$$

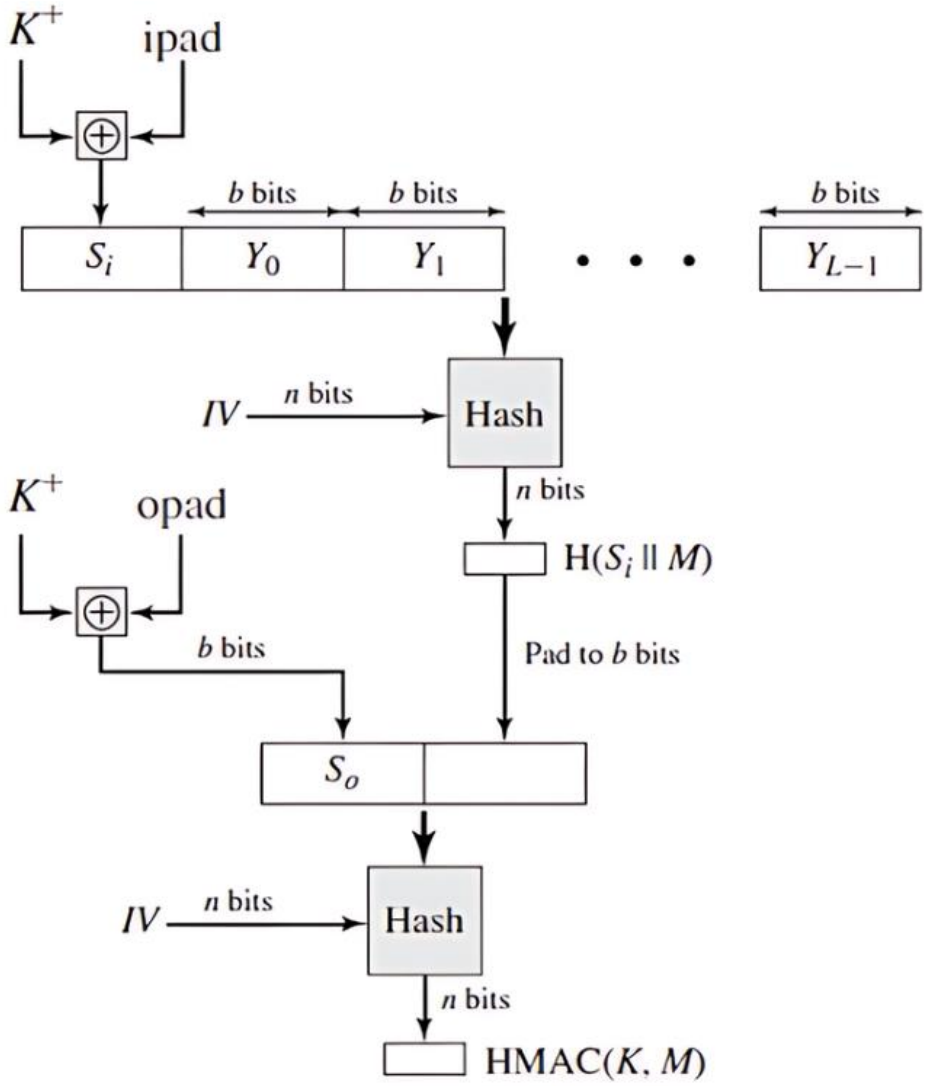
$$K' = \begin{cases} H(K), & |K| > block\_size \\ 0^*||K, & |K| < block\_size \\ K, & otherwise. \end{cases}$$

- $H(\cdot)$ is a hash function (e.g., MD5 and SHA-1).
- M is a plaintext message input to the HMAC.
- K is an original secret key string.
- K' is an 8-bit padded secret key to the block size.
- $ipad$ is an inner padding constant = 00110110 = 0x36
- $opad$ in an outer padding constant = 01011100 = 0x5C

Hash-based Message Authentication Code (HMAC)

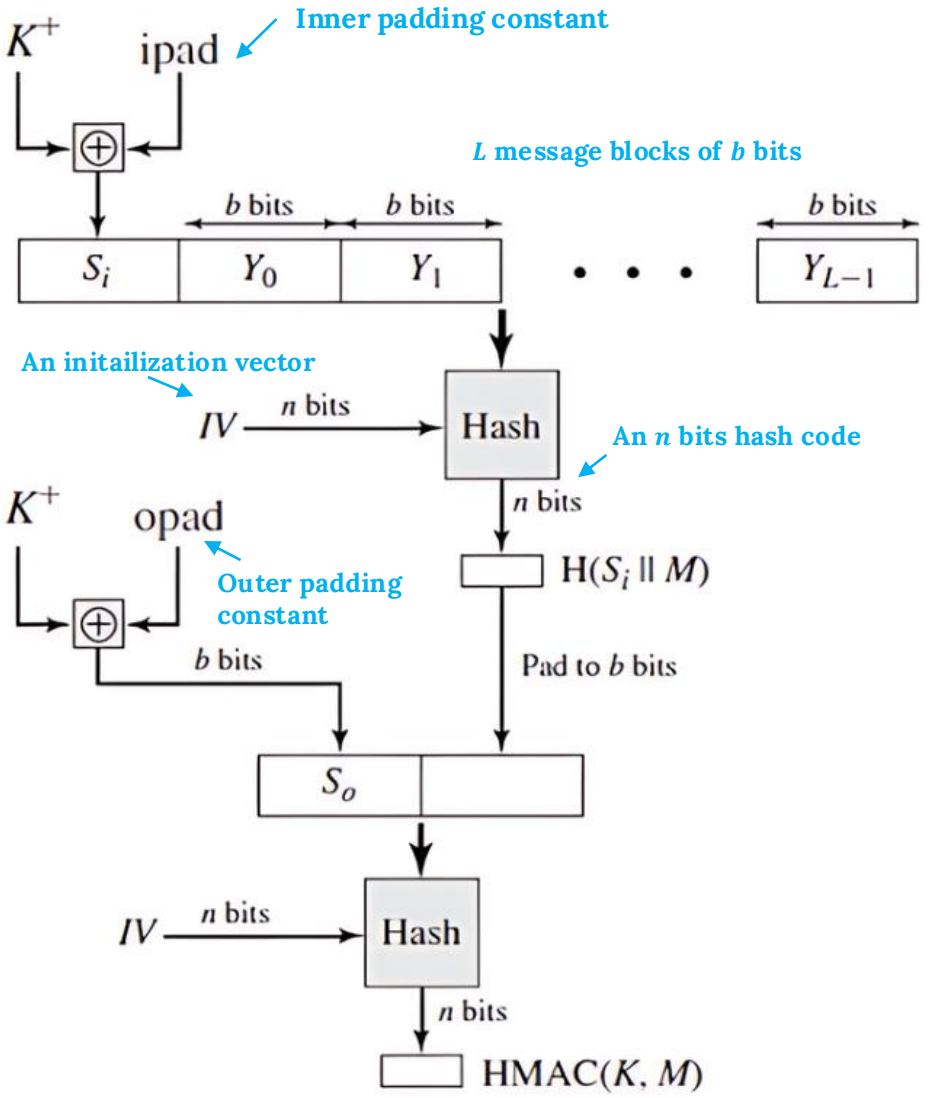
- **HMAC Algorithm:**

  - $H(\cdot)$ is a hash function (e.g., MD5 and SHA-1).
  - M is a plaintext message input to the HMAC.
  - b is the number of bits in a plaintext message block.
  - L is the number of plaintext message blocks.
  - Y_i represents the i-th plaintext block, where $0 \leq i \leq L-1$.
  - IV is an initialization value input to the hash function.
  - n is the number of bits in the hash code produced ($n < b$).
  - K is an original secret key string.
  - K' is an 8-bit padded secret key to the block size.
  - $ipad$ is an inner padding constant = 00110110 = 0x36
  - $opad$ in an outer padding constant = 01011100 = 0x5C



Hash-based Message Authentication Code (HMAC)

- **HMAC Algorithm:**
  1. Generate **a secret key block (K' or K^+)** from the original secret key (K) and XOR it with $ipad$ to produce an inner padding string S_i.
  2. Divide the plaintext message into L block of b bits.
     $$Y = \{Y_0, Y_1, \dots, Y_{L-1}\}$$
  3. Concatenate the inner padding string S_i to the plaintext message blocks Y i.e., $(S_i \,||Y)$.
  4. Pass the concatenated string to the selected hash function with IV as a key, resulting $H(S_i \,||Y)$
  5. XOR the secret key block (K') with the $opad$ to produce an outer padding string S_o.
  6. Concatenate the XORed result to the first hash value and pass through $H(\cdot)$ again, i.e., $A = H(S_o \,||H(S_i \,||Y))$.



Hash-based Message Authentication Code (MAC)

```
function hmac is
    input:
        key:        Bytes    // Array of bytes
        message:    Bytes    // Array of bytes to be hashed
        hash:       Function // The hash function to use (e.g. SHA-1)
        blockSize:  Integer  // The block size of the hash function (e.g. 64 bytes for SHA-1)
        outputSize: Integer  // The output size of the hash function (e.g. 20 bytes for SHA-1)

    // Compute the block sized key
    block_sized_key = computeBlockSizedKey(key, hash, blockSize)

    o_key_pad ← block_sized_key xor [0x5c blockSize]   // Outer padded key
    i_key_pad ← block_sized_key xor [0x36 blockSize]   // Inner padded key

    return  hash(o_key_pad ∥ hash(i_key_pad ∥ message))

function computeBlockSizedKey is
    input:
        key:        Bytes    // Array of bytes
        hash:       Function // The hash function to use (e.g. SHA-1)
        blockSize:  Integer  // The block size of the hash function (e.g. 64 bytes for SHA-1)

    // Keys longer than blockSize are shortened by hashing them
    if (length(key) > blockSize) then
        key = hash(key)

    // Keys shorter than blockSize are padded to blockSize by padding with zeros on the right
    if (length(key) < blockSize) then
        return  Pad(key, blockSize) // Pad key with zeros to make it blockSize bytes long

    return  key
```

Advantages and Disadvantages of HMAC

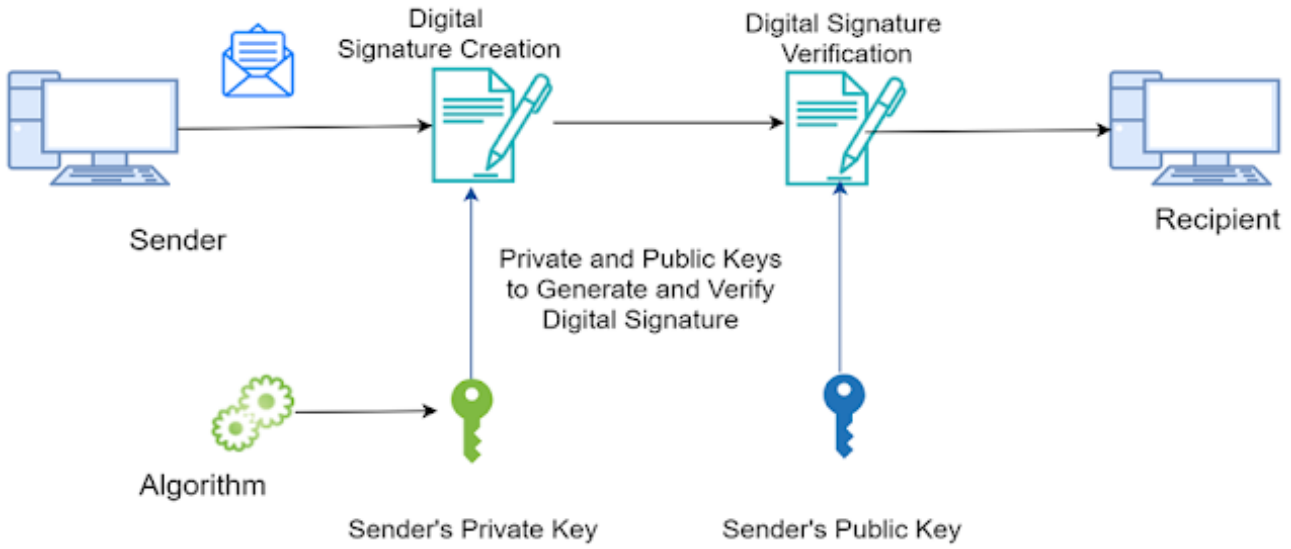
- **Advantages:**
  - **Efficient and Lightweight:** HMAC is small and fast, as it typically relies on a simple mathematical hash function, making it suitable for low-resource environments.
  - **No Need for Public Key Infrastructure:** It is particularly useful in scenarios where public key cryptography is restricted or not feasible.
  - **Resistant to Cryptanalysis:** HMAC is strong against various cryptographic attacks, as its security does not solely depend on the collision resistance of the underlying hash function
- **Disadvantages:**
  - **Lacks Receiver-Specific Confidentiality:** When multiple receivers share the same secret key, HMAC cannot guarantee the message was intended for a specific recipient, which can compromise confidentiality.
  - **Key Exchange Vulnerabilities:** HMAC still relies on secure key exchange mechanisms. Without proper protection, it can be vulnerable to replay attacks, such as those demonstrated in Man-in-the-Middle (MitM) scenarios like Darth's attack in Diffie-Hellman key exchange.

Section 5.3:

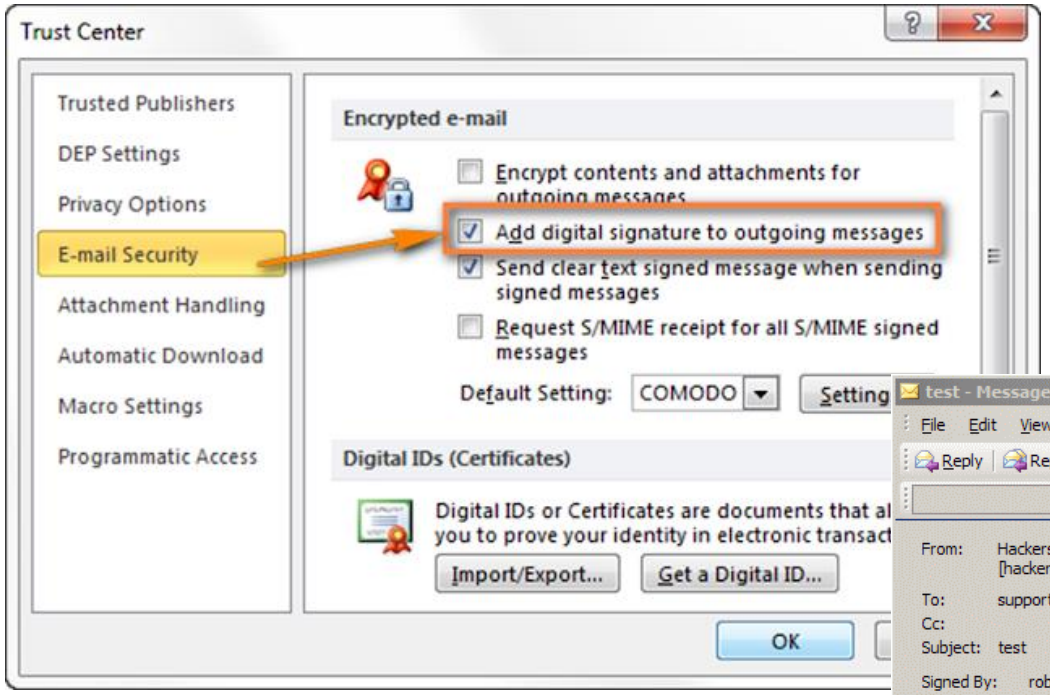
Digital Signatures

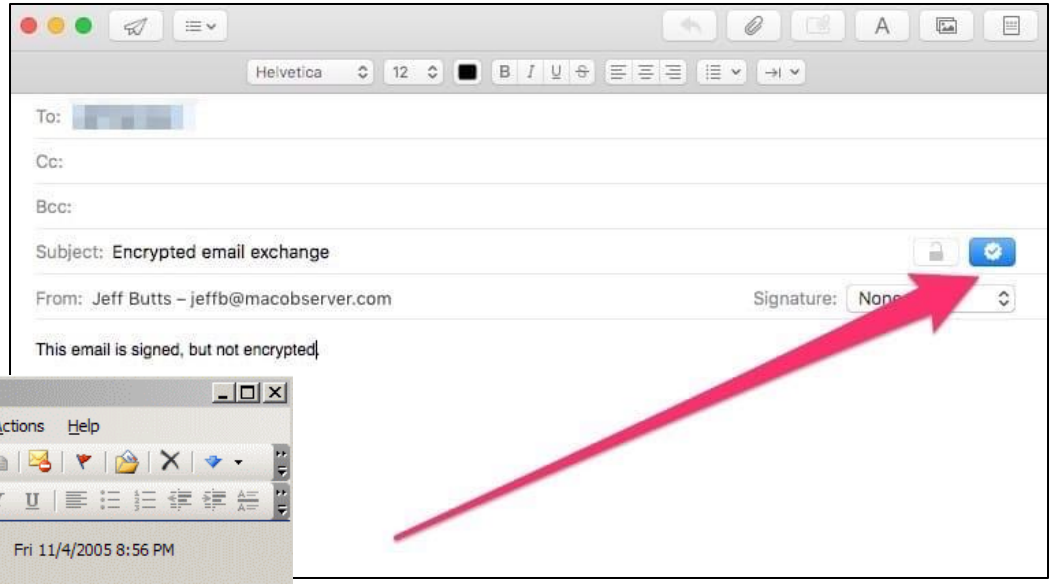
An Introduction to Digital Signatures

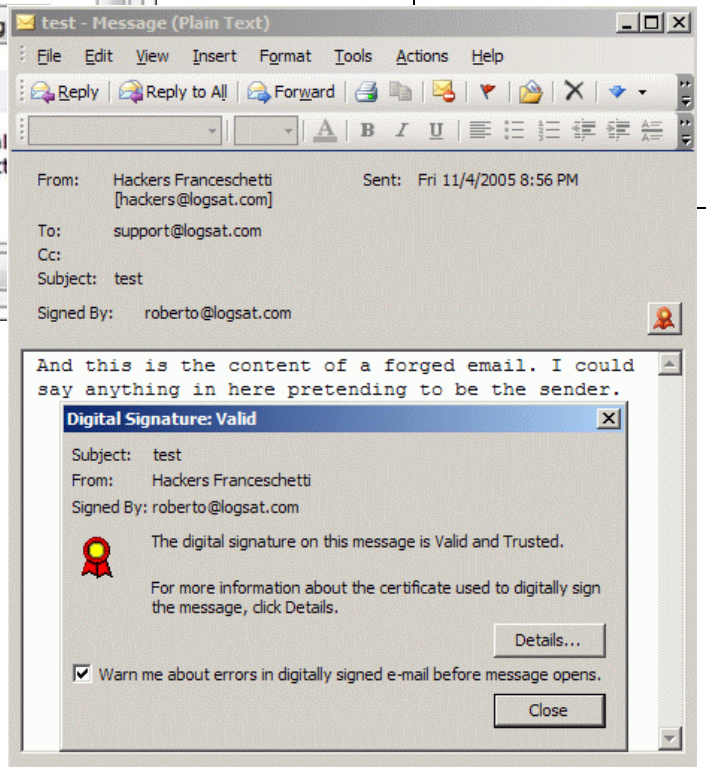
- **Digital Signature** is:
  - A mathematical technique used to verify the authenticity and integrity of a message or digital document.
  - An electronically generated signature created using cryptographic algorithms to confirm the identity of the sender and ensure that the content has not been altered during transmission.
- Digital signatures are a fundamental component of modern security protocols, including **email encryption**, **software distribution**, and **secure transactions**.



An Introduction to Digital Signatures







Key Concepts of Digital Signatures

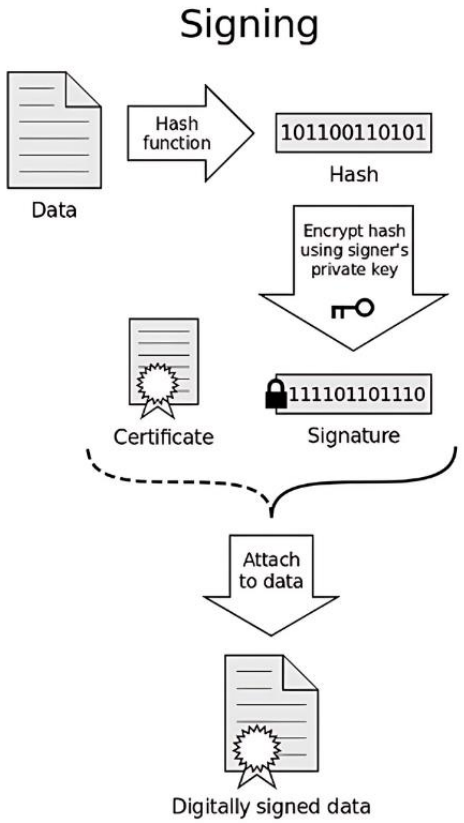
- Digital signatures leverage the benefits of public and private key infrastructure:
  - The **sender** uses **a signing key (i.e., the private key)** to generate *a digital signature* for the message. The sender then transmits both the message and its digital signature to the receiver.
  - The **receiver** uses **the verification key (i.e., the sender's public key)** to verify the authenticity and integrity of the message. This process ensures that:
    - The message originates from the claimed sender, and
    - The message has not been tampered with during transmission.
- **Key Properties of Digital Signatures:**
  - The identity of the sender and the date/time the signature was created must be **verifiable**.
  - The contents of the message or document at the time of signing must be **authenticatable**, ensuring they have not been altered.
  - The signature must be **verifiable** by trusted third parties to allow for independent validation and dispute resolution.

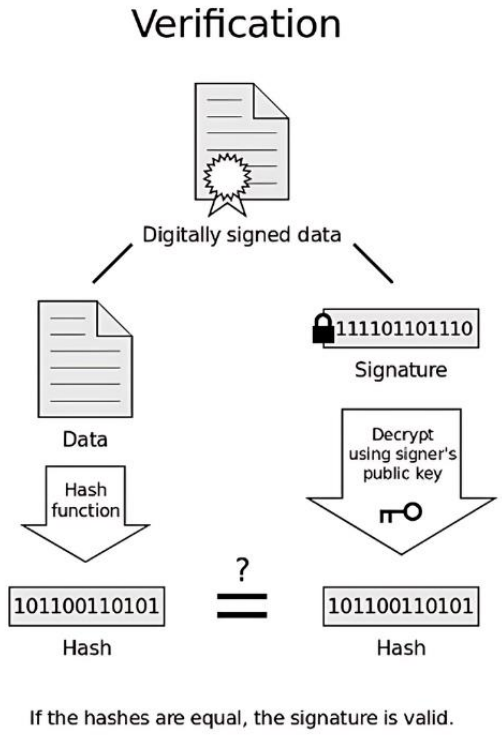
How Digital Signatures Work

1. **Key Generation:** A key generation algorithm creates a private key and its corresponding public key.
2. **Signing:** A signing algorithm takes the message and the private key as inputs and generates a digital signature.
3. **Verification:** A verification algorithm uses the message and the sender's public key to validate the signature, confirming the message's authenticity and integrity.

Various Digital Signature Schemes Can Be Used:

- Elgamal Digital Signature Scheme
- Schnorr Digital Signature Scheme
- Digital Signature Standard (DSS) and Digital Signature Algorithm (DSA)





Elgamal Digital Signature Scheme

- **Elgamal digital signature scheme** is based on the **computational hardness of discrete logarithms** over **a large prime number** q.
- Generating a signature involves computing a pair (S_1, S_2), which is **computationally intensive** compared to simpler schemes like RSA.
- It provides **authenticity and integrity**, ensuring that a message M was indeed **sent by the claimed sender** and **verified by a specific recipient**.
- The scheme relies on two **global parameters**: **a large prime number** q and **its primitive root** α, both of which are publicly known.

$M = 14$

Elgamal Digital Signature Scheme

- **Step 1: Public and Private Key Pair Generation:**
  - **1.1 Select Initial Parameters:** Choose **a large prime number** p and **its primitive root** α.

    Example: $p = 19, \alpha = 10$
  - **1.2 Generate a Private Key SK_A:** Select **a random integer** for **a private key** SK_A such that $1 < SK_A < p - 1$.

    Example: $SK_A = 7$, since $1 < 7 < 18$
  - **1.3 Compute the Public Key PK_A:** Use the formula: $PK_A = \alpha^{SK_A} \bmod p$

    Example: $PK_A = 10^7 \bmod 19 = 10{,}000{,}000 \bmod 19 = 15$
  - **1.4 Finalize the Key Pair:** The resulting **private and public key pair** is: $(SK_A, (p, \alpha, PK_A))$

    Example: $(7, (19,10,15))$

$M = 14$

$(SK_A, (p, \alpha, PK_A)) = (7, (19, 10, 15))$

Elgamal Digital Signature Scheme

- **Step 2: Create a Signature for a Given Message:**
  - **2.1** Select a random integer K that is relatively prime to $p - 1$; that is, $GCD(K, p - 1) = 1$ and $1 \leq K \leq p - 1$.

    Example: $K = 5$ because $GCD(5, 18) = 1$ and $1 \leq 5 \leq 18$.

  - **2.2** Compute $S_1 = \alpha^K \bmod p$.

    Example: $S_1 = 10^5 \bmod 19 = 100{,}000 \bmod 19 = 3$.

  - **2.3** Compute $S_2 = K^{-1}(M - SK_A S_1) \bmod (p - 1)$, where $K^{-1} = K^{(p-1)-1} \bmod (p - 1)$ is the modular inverse of K.

    Example: $K^{-1} = 5^{17} \bmod 18 = 11$ and $S_2 = 11(14 - 7(3)) \bmod 18 = 13$.

  - **2.4** Form the signature (S_1, S_2).

    Example: $(S_1, S_2) = (3, 13)$.

$M = 14$

Elgamal Digital Signature Scheme

$(SK_A, (p, \alpha, PK_A)) = (7, (19, 10, 15))$

$(S_1 . S_2) = (3, 13)$

- **Step 3: Verify the Authenticity of a Received Message:**
  - **3.1** Compute a verification value from the received message: $V_1 = \alpha^M \bmod p$

    Example: $V_1 = 10^{14} \bmod 19 = 16$

  - **3.2** Compute the verification value from the received signature: $V_2 = PK_A^{S_1} S_1^{S_2} \bmod p$

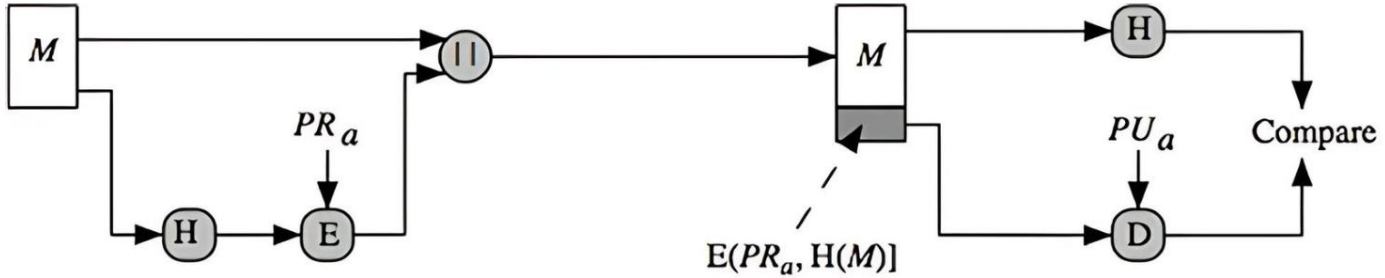
    Example: $V_2 = 15^3 \cdot 3^{13} \bmod 19 = 5{,}380{,}840{,}125 \bmod 19 = 16$

  - **3.3** Compare the two verification values V_1 and V_2.

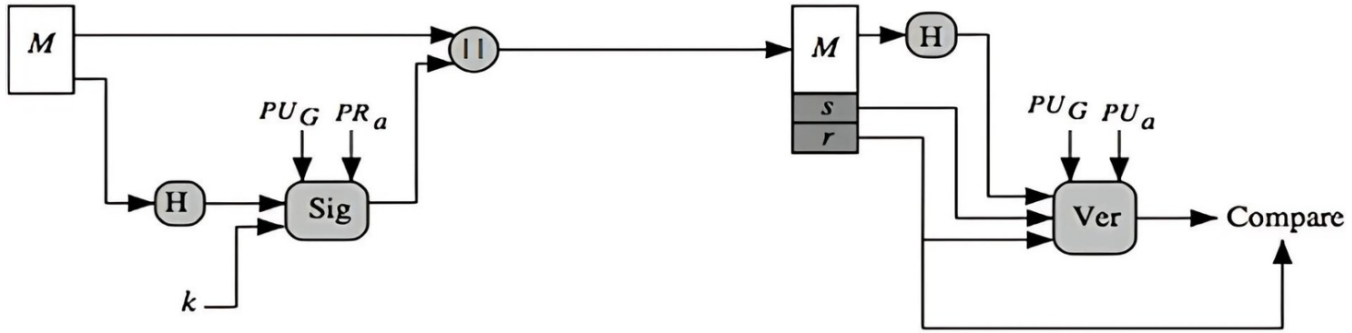
    Example: In this case, $V_1 = V_2 \sim 16 = 16$, indicating that the message M is authentic.

Digital Signature Standard (DSS) and Algorithm (DSA)

- **NIST** has published the Federal Information Processing Standard 186, known as **the Digital Signature Standard (DSS)**.
- The DSS makes use of the SHA and presents a new digital signature technique, the **Digital Signature Algorithm (DSA)**.
- Latest version also incorporates digital signature algorithms based on **RSA and on elliptic curve cryptography**.

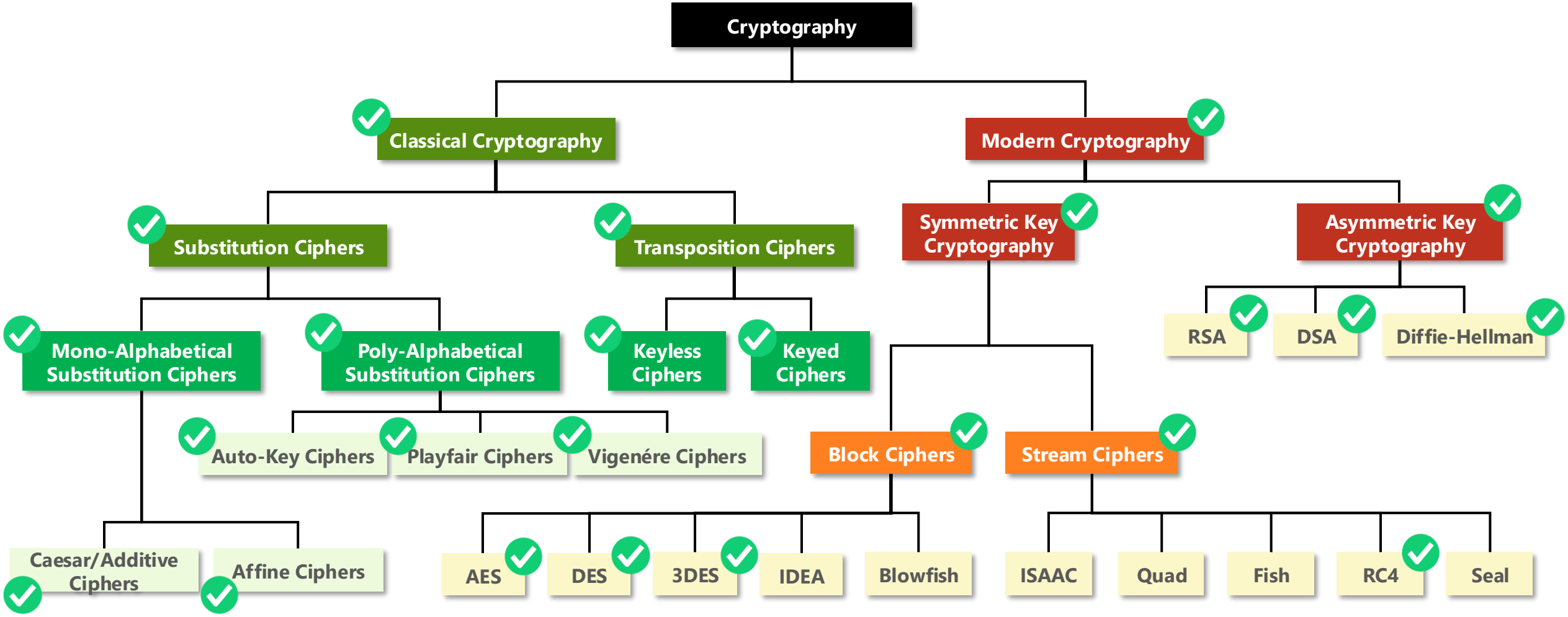


The RSA Approach + Elliptic Curve Cryptography



The DSS Approach (i.e., DSA)

Classification of Cryptographic Techniques



End of the Lecture

Please don’t hesitate to raise your hand and ask questions if you're curious about anything!