Q1. Define Symmetric Cipher Model

Answer:

- The **symmetric cipher model** is a framework used to describe **conventional (secret-key) encryption** systems.
- It involves **one shared secret key** used for both **encryption and decryption** of messages.
- The model has these main components:
  1. **Plaintext (input message)**
  2. **Encryption algorithm** – transforms plaintext into ciphertext using the key
  3. **Secret key** – shared between sender and receiver
  4. **Ciphertext (output encrypted message)**
  5. **Decryption algorithm** – reverses the encryption using the same key
- **Requirement:** Both sender and receiver must **securely obtain and keep the key secret.**

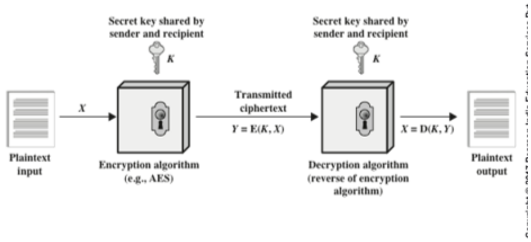

Figure 3.1 Simplified Model of Symmetric Encryption

📘 Q2 — Requirements for Secure Use of Conventional Encryption

From your Chapter 3 file Chapter 3 INS :

To use conventional (symmetric key) encryption securely, **two main requirements** must be satisfied:

1. **A strong encryption algorithm**
   - The algorithm used for encryption and decryption must be computationally secure and resistant to cryptanalysis or brute-force attacks.
   - Even if the attacker knows the algorithm, it should still be infeasible to break without the key.
2. **Secure key distribution and secrecy**
   - Both the sender and the receiver must obtain copies of the **secret key** in a secure way.
   - They must also **keep the key secret** from all third parties.
   - If the key is leaked, the security of the entire system is lost.

📘 Q3 — Cryptographic Systems (Three Independent Dimensions)

From your Chapter 3 file Chapter 3 INS :

Cryptographic systems can be **characterized along three independent dimensions:**

---

1. Type of Operations Used

- Refers to the **mathematical operations** used to transform plaintext into ciphertext.
- Two basic types:
  - **Substitution** — Each element of plaintext is replaced by another element.
  - **Transposition (permutation)** — The positions of plaintext elements are changed.
- Most systems actually use a **combination** of both for stronger security.

---

2. Number of Keys Used

- This determines whether the system is **symmetric (single-key)** or **asymmetric (public-key).**
- **Symmetric (conventional):** Same key is used for encryption and decryption.
- **Asymmetric (public-key):** One key (public) is used for encryption, and a different key (private) is used for decryption.

---

3. Way in Which the Plaintext is Processed

- Refers to how the plaintext data is **handled during encryption.**
- Two main approaches:
  - **Block cipher:** Processes plaintext in fixed-size blocks.
  - **Stream cipher:** Processes plaintext continuously one bit or byte at a time.

✅ **Summary Table (Optional to draw):**

Dimension	Type	Example
Operations	Substitution / Transposition	Caesar cipher / Rail fence
Keys used	Symmetric / Asymmetric	DES / RSA
Processing	Block / Stream	DES / RC4

📘 Q4 — Cryptanalysis and Brute-Force Attack

From your Chapter 3 file Chapter 3 INS :

🔍 Cryptanalysis

- The **science (or art) of breaking ciphers** and finding the plaintext or key without prior knowledge of the key.
- It uses **analytical techniques** and **knowledge of the algorithm, statistical properties** of the plaintext, and **patterns in the ciphertext.**
- Goal: **Deduce the key or plaintext** without having the key.

📝 **Also called: Code breaking or cipher breaking.**

💥 Brute-Force Attack

- A method that tries **every possible key** until the correct one is found.
- It **does not use analysis**, just **exhaustive trial and error.**
- Works because symmetric keys are usually from a **finite keyspace.**
- Becomes **impractical if the keyspace is very large**, because it takes too long.

✅ **Key Difference:**

Cryptanalysis	Brute-force
Uses logic, statistics, and structure of cipher	Tries all possible keys blindly
Often faster if cipher is weak	Guaranteed to work, but can be very slow

📘 Q5 — Encryption Scheme Security

From your Chapter 3 file Chapter 3 INS :

Encryption schemes can be classified based on their **level of security**:

🔐 1. Unconditionally Secure

- No matter how much **time or computational power** an attacker has, they **cannot decrypt the ciphertext** because
  there isn't enough information in the ciphertext to deduce the plaintext.
- Even with **infinite resources**, the cipher cannot be broken.
- Example: **One-Time Pad** (the only known perfectly secure system).

🧠 2. Computationally Secure

- It is **theoretically possible** to break the cipher, but:
  - The **cost of breaking it exceeds the value of the encrypted information**, or
  - The **time required to break it exceeds the useful lifetime of the information**.
- Most modern encryption systems aim for **computational security**.

✅ Summary:

Type	Meaning	Practicality
Unconditionally secure	Impossible to break even with infinite resources	Rare (One-Time Pad)
Computationally secure	Possible but not feasible due to time/cost	Used in practice

📘 Q6 — Substitution Technique

From your Chapter 3 file Chapter 3 INS :

✍️ Definition

- A **substitution technique** is an encryption method in which the **letters (or bits) of the plaintext are replaced by other letters, numbers, or symbols.**
- If plaintext is seen as a **sequence of bits**, substitution means **replacing each plaintext bit pattern with a ciphertext bit pattern.**

📌 Key Points

- Each element (character or bit group) in the plaintext is **mapped to another element.**
- The **positions stay the same** — only the **values change.**
- Substitution can be applied at:
  - **Letter level** (Caesar cipher)
  - **Pair of letters (digrams)** (Playfair cipher)
  - **Block of letters** (Hill cipher)
- Main goal: **Confuse the relationship** between the ciphertext and the key (called **confusion** by Shannon).

✅ Example

- **Caesar Cipher**: Replaces each letter by another letter a fixed number of positions down the alphabet.
  - `plain: meet` → `cipher: PHHW` (shift of 3)

📘 Q7 — Caesar Cipher

From your Chapter 3 file Chapter 3 INS :

📝 Definition

- The **Caesar cipher** is the **simplest and earliest known substitution cipher**, used by **Julius Caesar**.
- It works by **shifting each letter in the plaintext by a fixed number of positions (k)** down the alphabet.
- The alphabet is **wrapped around** so that after **Z comes A**.

⚙️ Working

Example (Shift = 3):

vbnet Copy code

```
Plain:     meet me after the toga party
Cipher:    PHHW PH DIWHU WKH WRJD SDUWB
```

Letter mapping:

mathematica Copy code

```
Plain:  a b c d e f g h i j k l m n o p q r s t u v w x y z
Number: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Shift:  D E F G H I J K L M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z  A  B  C
```

Mathematical form:

- **Encryption:**

$$C = E_k(p) = (p + k) \mod 26$$

- **Decryption:**

$$p = D_k(C) = (C - k) \mod 26$$

where

- p = plaintext letter number
- k = key (shift) value
- C = ciphertext letter number

✅ **Key point:** It is easy to understand but **very weak** — only **25 possible keys**, so **easy to brute-force**.

📘 Q8 — Monoalphabetic Cipher

From your Chapter 3 file Chapter 3 INS :

📝 Definition

- A **monoalphabetic cipher** is a substitution cipher that uses **only one fixed substitution alphabet** for the entire message.
- Each plaintext letter is **always replaced by the same ciphertext letter** throughout the message.

⚙️ Key Points

- The **cipher alphabet** can be **any random permutation of the 26 letters**.
- Number of possible keys = **26! (factorial)** → greater than **4 × 10^{26} possible keys**, which is **much larger than DES key space**.
- Because each letter is consistently mapped, it **preserves the frequency pattern** of letters from the plaintext.

⚠️ Weakness

- Even though there are many keys, it is **easy to break using frequency analysis**.
- Attackers analyze **most common letters (like E, T, A)**, **digrams (th)** and **trigrams (the)** to crack it.

✅ Key terms

- **Homophones:** Using multiple symbols for one letter to reduce frequency clues.
- **Digrams:** 2-letter combinations (e.g. *th*)
- **Trigrams:** 3-letter combinations (e.g. *the*)

📘 Q9 — Playfair Cipher

From your Chapter 3 file Chapter 3 INS :

📝 Definition

- The **Playfair cipher** is a **multiple-letter (digram) substitution cipher.**
- It encrypts **pairs of letters (digrams)** together, rather than single letters.
- Invented by **Sir Charles Wheatstone in 1854**, and used by the **British Army in World War I** and **Allied forces in World War II.**

⚙️ Construction of the 5×5 Key Matrix

1. **Choose a keyword** (e.g. `MONARCHY`).
2. Write the keyword (without duplicates) left to right, top to bottom in a 5×5 matrix.
3. Fill remaining spaces with unused letters of the alphabet in order (combine I/J in one cell).

Example:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

⚙️ Encryption Rules

Break plaintext into **pairs of letters (digrams).**
For each pair:

1. **Same Row:** Replace each with the letter **to its right** (wrap around to the start if needed).
2. **Same Column:** Replace each with the letter **below it** (wrap around to the top if needed).
3. **Rectangle Rule:** If in **different rows and columns**, replace each with the letter in the **same row but in the other letter's column.**

⚠️ If both letters are the same (like "LL"), insert a filler (X) between them.

✅ Key Points

- Encrypts 2 letters at a time → hides letter freque[illegible]s better than simple substitution.
- Much stronger than monoalphabetic ciphers, but still breakable with enough known plaintext.

📘 Q10 — Hill Cipher

From your Chapter 3 file Chapter 3 INS **:**

📝 Definition

- The **Hill cipher** is a **polyalphabetic substitution cipher based on linear algebra**, invented by **Lester Hill in 1929.**
- It encrypts **blocks (vectors) of letters using matrix multiplication**, which hides individual letter frequencies.

⚙️ Working

1. **Choose an n × n key matrix** (n = block size, usually 2 or 3).
2. Convert each plaintext letter to a number (A = 0, B = 1, ... Z = 25).
3. Break plaintext into **blocks of n letters.**
4. Multiply each plaintext vector by the key matrix (mod 26) to get the ciphertext vector.
5. Convert the resulting numbers back to letters.

Example (Concept):

$$C = K \times P \pmod{26}$$

where

- C = ciphertext vector
- P = plaintext vector
- K = key matrix

Decryption:

$$P = K^{-1} \times C \pmod{26}$$

where K^{-1} is the modular inverse of the key matrix.

⚡ Features

- **Hides frequency of individual letters** (unlike monoalphabetic ciphers).
- Larger matrix → hides digram/trigram frequencies too.
- **Strong against ciphertext-only attack**
- But **vulnerable to known-plaintext attacks** (if enough plaintext-ciphertext pairs are known, key can be solved using linear algebra).

📘 Q11 — Polyalphabetic Ciphers

(Example: **Vigenère Cipher**)

Definition:

- Polyalphabetic ciphers use **multiple substitution alphabets.**
- **A key determines which alphabet** is used at each letter position.
- Because the same plaintext letter can be encrypted differently depending on its position, **frequency analysis becomes difficult.**

⚡ Example: Vigenère Cipher

- Uses **26 Caesar cipher alphabets** with shifts from 0 to 25.
- **A repeating keyword** controls which shift is applied to each letter.

Example:

makefile

```
Key:        deceptive deceptive deceptive
Plaintext:  wearediscoveredsaveyourself
Ciphertext: ZICVTWQNGRZGVTWAVZHCQYGLMGJ
```

💻 Algorithm / Pseudocode

Convert: A=0 ... Z=25

Encryption:

vbnet

```
Input: plaintext[], key[] (repeated to match length of plaintext)
Output: ciphertext[]

for i from 0 to length-1:
    p = plaintext[i]
    k = key[i]
    c = (p + k) mod 26
    ciphertext[i] = c
```

Decryption:

vbnet

```
for i from 0 to length-1:
    c = ciphertext[i]
    k = key[i]
    p = (c - k + 26) mod 26
    plaintext[i] = p
```

📘 Q12 — Vernam Cipher / One-Time Pad

From your Chapter 3 file Chapter 3 INS :

📝 Definition

- The **Vernam cipher** is a **type of polyalphabetic cipher** where each plaintext character is combined with a **random key character** of the same length.
- The **One-Time Pad** is an **improved version proposed by Joseph Mauborgne**, which:
  - uses a **truly random key** that is as long as the message,
  - is used **only once** and then discarded.

⚡ Features

- **Unbreakable** if used correctly.
- Produces **completely random ciphertext.**
- Achieves **perfect secrecy** (no information about plaintext is leaked).

Problems:

- Requires generating huge amounts of random keys.
- Difficult key distribution: both sender and receiver must securely share a key of equal length to the message.

💻 Vernam Cipher Algorithm (Pseudocode)

Assume letters are converted to numbers (A=0...Z=25):

```vbnet
Input: plaintext[], key[]  (key length = plaintext length)
Output: ciphertext[]

for i from 0 to length-1:
    ciphertext[i] = (plaintext[i] + key[i]) mod 26
```

Decryption:

```vbnet
for i from 0 to length-1:
    plaintext[i] = (ciphertext[i] - key[i]) mod 26
```

📘 Q13 — Rail Fence Cipher

From your Chapter 3 file Chapter 3 INS **:**

📝 Definition

- The **rail fence cipher** is the **simplest transposition cipher.**
- It **rearranges the order of letters (no substitution).**
- Plaintext is written **diagonally on successive "rails" (rows),** then read row by row to form ciphertext.

⚡ Example (Depth = 2)

Plaintext:

```vbnet
meet me after the toga party
```

Step 1 – Write diagonally on 2 rails:

```css
m e m a t r h t g p r y
  e t e f e t e o a a t
```

Step 2 – Read row by row:

```makefile
Ciphertext: MEMATRHTGPRYETEFETEOAAT
```

💻 Algorithm / Pseudocode

Encryption:

```sql
Input: plaintext, depth (number of rails)
Output: ciphertext

1. Create an empty 2D array rails[depth][len(plaintext)]
2. direction = down
3. row = 0

4. For each character ch in plaintext:
       rails[row].append(ch)
       if row == 0 then direction = down
       if row == depth-1 then direction = up
       if direction == down then row++ else row--

5. Read the rails row by row to form ciphertext
```

Decryption:

- Place letters row by row in zig-zag pattern positions,
- then read them diagonally to reconstruct the plaintext.

✅ **Key Point:** Rail fence changes only the **order (position)** of letters → **transposition cipher**, not substitution.

Q14 — Row Transposition Cipher

Definition

- A **row transposition cipher** is a **transposition (permutation) cipher.**
- The plaintext is written **row by row** inside a rectangle of fixed column size.
- Then the letters are read **column by column in an order determined by a numeric key.**

Steps (Encryption)

1. Choose a **numeric key** that shows the **order of columns** (example: `4 3 1 2 5 6 7`).
2. Write the **plaintext row-wise** in a grid having **columns = key length.**
3. **Number the columns according to the key.**
4. Read the letters **column by column in the order of the key numbers** to get the **ciphertext.**

Pseudocode

vbnet

```
Input: plaintext, key[]
Output: ciphertext

cols = length(key)
rows = ceil(length(plaintext)/cols)
grid = fill plaintext row by row

ciphertext = ""
for k in sorted(key):           # column order as per key
    colIndex = position of k in key[]
    for r in 0..rows-1:
        if grid[r][colIndex] exists:
            ciphertext += grid[r][colIndex]
```

Example

Key:

```
4   3   1   2   5   6   7
```

Plaintext (written in rows):

css

```
a   t   t   a   c   k   p
o   s   t   p   o   n   e
d   u   n   t   i   l   t
w   o   a   m   x   y   z
```

Column order (according to key):

`4 → 3 → 1 → 2 → 5 → 6 → 7`

Ciphertext:

nginx

```
TTNAAPTMTSUOAODWCOIXKNLYPETZ
```

Key Point:

- This cipher **changes the order (position) of letters, not the letters themselves.**
- So it is a **transposition cipher**, not a substitution cipher.

📗 Q15 — Difference Between Stream Cipher and Block Cipher

From your Chapter 4 file Chapter 4 INS :

📝 Definition

Stream Cipher

- Encrypts **one bit or byte at a time.**
- Processes data as a **continuous stream.**

Block Cipher

- Encrypts **fixed-size blocks of plaintext** (e.g. 64-bit or 128-bit blocks).
- Processes data **block by block.**

📌 Difference Table

Feature	Stream Cipher	Block Cipher
Processing unit	One bit or byte at a time	A block (e.g. 64/128 bits) at a time
Speed	Faster for real-time applications	Slightly slower
Error propagation	Error affects only one bit/byte	Error can affect the entire block
Memory requirement	Low (processes small pieces)	Higher (needs to store entire block)
Common examples	RC4	DES, AES
Use cases	Streaming audio/video, network data	File encryption, database encryption

---

✅ Key Point:

- **Stream ciphers** are used for **real-time data transmission.**
- **Block ciphers** are used for **secure storage and bulk data encryption.**

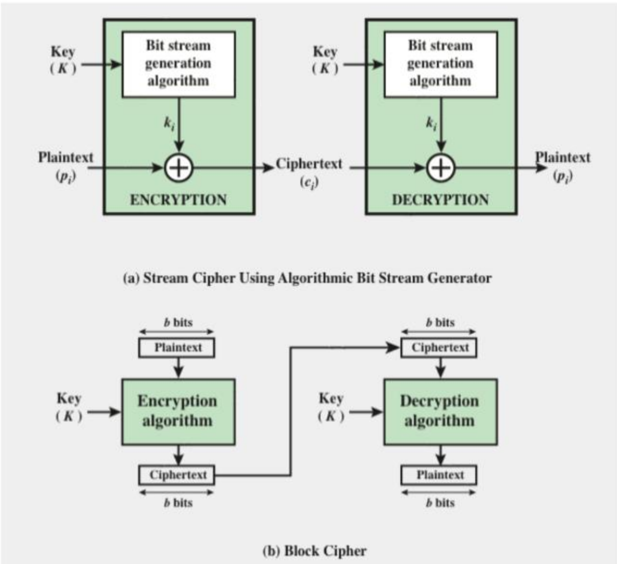

Figure 4.1 Stream Cipher and Block Cipher

📗 Q16 — Motivation for Feistel Cipher

From your Chapter 4 file Chapter 4 INS :

📝 Background

- Claude Shannon (founder of modern cryptography) said a good cipher should use:
  - **Confusion** — make the relationship between key and ciphertext as complex as possible.
  - **Diffusion** — spread the influence of one plaintext symbol over many ciphertext symbols.
- But building a cipher that is both **secure and efficient** is hard.

⚡ Feistel's Idea

- **Horst Feistel** proposed a **practical way** to build strong block ciphers from simple components.

Motivations:

1. **Combine confusion and diffusion** in an easy-to-implement structure.
2. **Make encryption and decryption use the same structure**, just reversing the key order — reduces design complexity.
3. **Allow building strong ciphers from weak functions** by repeating them in multiple rounds.
4. **Simplify analysis** — easy to prove and study the security mathematically.
5. **Flexibility** — can vary block size, key size, number of rounds, and round function design.

✅ **Key Point:**

- The **Feistel structure** is used in many real ciphers (DES, 3DES, etc.) because it allows **strong encryption with efficient hardware/software implementation.**

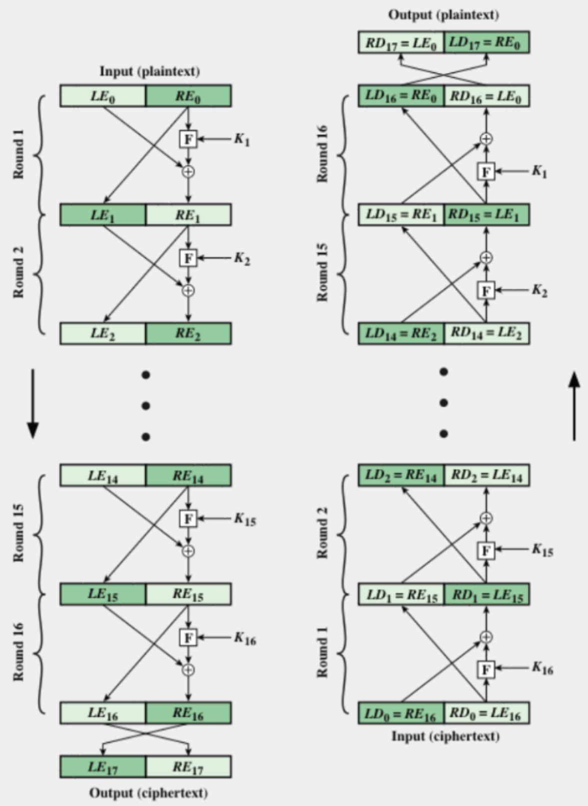

Figure 4.3 Feistel Encryption and Decryption (16 rounds)