Block Ciphers & Public Key Cryptography

Q. Describe the different modes of operation for block ciphers (ECB, CBC, CFB, OFB, CTR).

Block ciphers (like AES, DES) encrypt **fixed-size blocks** of data (e.g., 64-bit or 128-bit). However, real-world messages are often longer than one block or not evenly divisible.

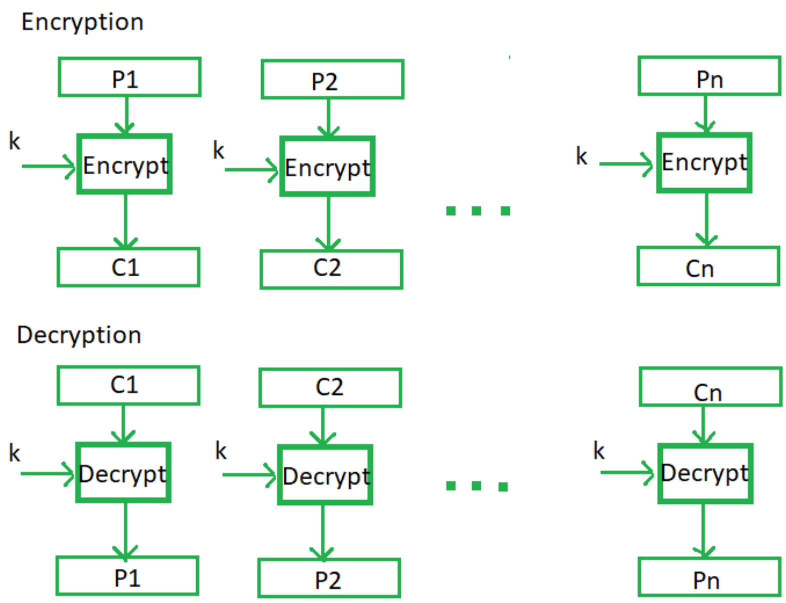
To handle this, **block cipher modes of operation** define **how to apply encryption to data larger than a single block**, and also introduce features like **randomness**, **feedback**, and **parallelism**.

Common Modes of Operation

1. ECB (Electronic Codebook Mode)

- **How it works**: Encrypts each block **independently** using the same key.
- **Formula**: $C_i = E(K, P_i)$
- **Issue**: Identical plaintext blocks produce identical ciphertext blocks.
- **Use Case**: Small, fixed-size data like encrypting keys.

Simple and parallelizable
Not secure for long or structured data (reveals patterns)

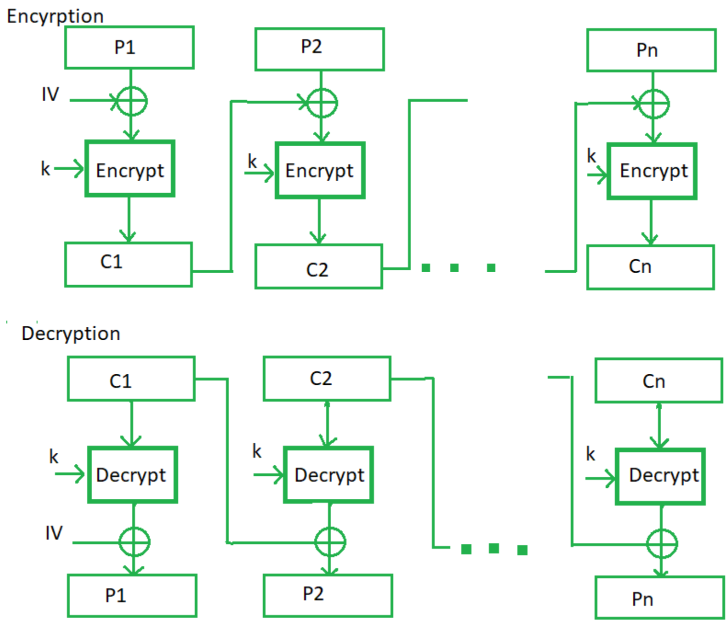


2. CBC (Cipher Block Chaining Mode)

- **How it works**: XOR each plaintext block with the **previous ciphertext block**, then encrypt.

- **First block** uses an **Initialization Vector (IV)**.
- **Formula**: $C_i = E(K, P_i \oplus C_{i-1})$
  $C_0 = E(K, P_0 \oplus IV)$
- **Decryption**: $P_i = D(K, C_i) \oplus C_{i-1}$

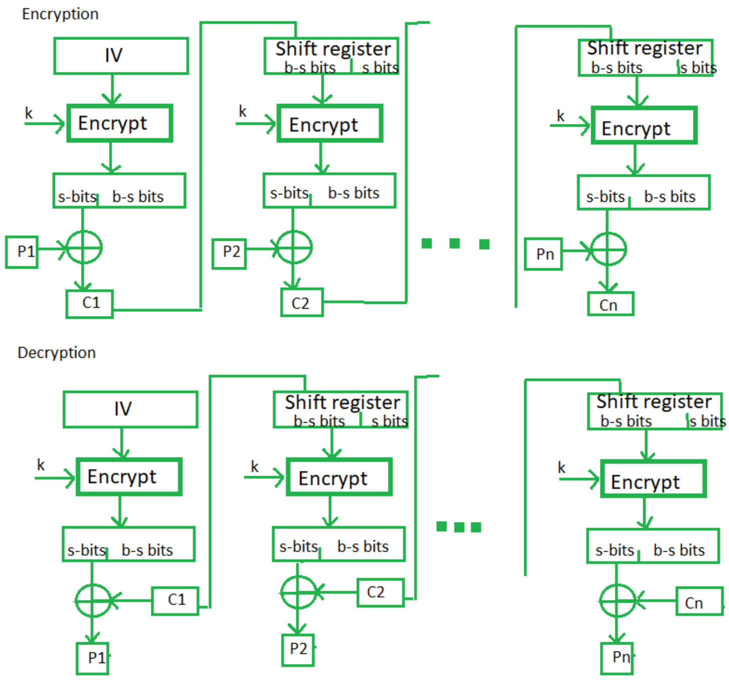
More secure than ECB
Not parallelizable (must process sequentially)



3. CFB (Cipher Feedback Mode)

- **How it works**: Turns a block cipher into a **self-synchronizing stream cipher**.
- Encrypts the **previous ciphertext block or IV**, then XORs it with the plaintext.
- **Formula**:
  $C_i = P_i \oplus E(K, C_{i-1})$
  $C_0 = P_0 \oplus E(K, IV)$

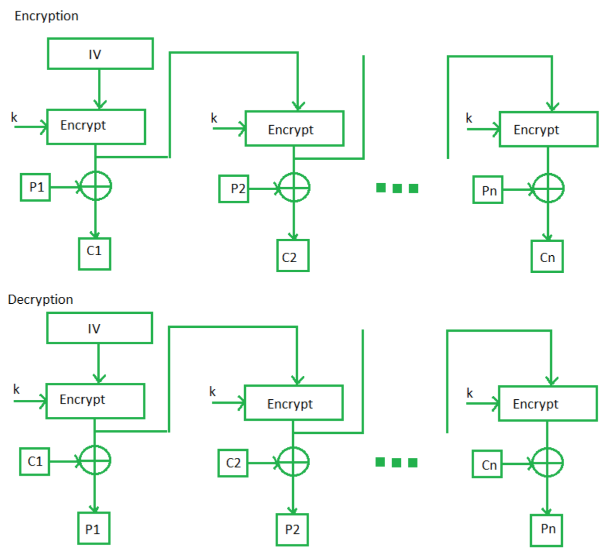
Encrypts smaller units (bits or bytes)
Errors propagate to next block



4. OFB (Output Feedback Mode)

- **How it works**: Encrypts the **output of the previous encryption**, then XORs with plaintext.
- **Does not use ciphertext as feedback**, only keystream.
- **Formula**:
  $Output_0 = E(K, IV)$
  $Output_i = E(K, Output_{i-1})$
  $C_i = P_i \oplus Output_i$

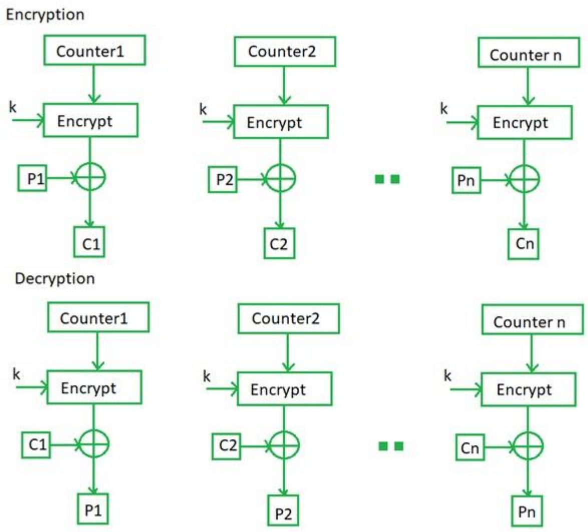
Error doesn't propagate
Precompute keystream for speed
Vulnerable if keystream reused



5. CTR (Counter Mode)

- **How it works**: Converts a block cipher into a **stream cipher** using a **counter** value.
- Encrypts counter + nonce and XORs with plaintext.
- **Formula**:
  $C_i = P_i \oplus E(K, Nonce \| Counter_i)$

Highly parallelizable (encryption/decryption)
Fast and secure
Nonce must never be reused



Q. Explain the structure and working of the DES algorithm.

DES (Data Encryption Standard) is a **symmetric key block cipher** developed in the 1970s by IBM and adopted as a standard by the U.S. government.

- It encrypts **64-bit blocks** of plaintext using a **56-bit key** (8 extra bits are used for parity, not encryption).
- DES uses a **Feistel structure** with **16 rounds** of encryption.

General Overview

- **Type**: Symmetric key block cipher (same key for encryption and decryption).
- **Block Size**: 64 bits (plain text is processed in blocks of 64 bits).
- **Key Size**: 64 bits, but only 56 bits are used for encryption (every 8th bit is discarded for parity).
- **Output**: 64-bit ciphertext.

- **Number of Rounds**: 16.

DES Process Flow

Step 1: Initial Permutation (IP)

- The 64-bit plaintext undergoes a fixed permutation (bit shuffling).
- Example from the IP table: bit positions might be rearranged like 58, 50, 42, ..., 7.
- Output is still 64 bits, split into:
  - **LPT (Left Plain Text)** – 32 bits
  - **RPT (Right Plain Text)** – 32 bits

Step 2: Key Generation (Key Transformation)

- **Input**: 64-bit key → remove parity bits → 56-bit effective key.
- **Process**:
  - Split into two 28-bit halves.
  - Left circular shift (by 1 or 2 bits depending on the round):
    - **Rounds 1, 2, 9, 16** → shift by 1 bit
    - **Other rounds** → shift by 2 bits
  - Combine and compress to form a **48-bit round key** using a predefined table.
- **Output**: 16 unique 48-bit subkeys (one for each round).

Step 3: 16 Rounds of DES Operations

Each round performs the following:

a. Expansion Permutation

- 32-bit RPT is expanded to 48 bits using a predefined expansion table.
- It splits RPT into 8 blocks of 4 bits, each expanded to 6 bits by repeating edge bits (padding 2 bits).

b. XOR Operation

- Expanded RPT (48-bit) is XORed with the current 48-bit round key.

c. Substitution (S-boxes)

- XOR result is split into 8 blocks of 6 bits each.
- Each 6-bit block enters a corresponding **S-box**.

- Each S-box:
    - Takes 6-bit input.
    - Uses first and last bit to determine the row.
    - Uses the middle 4 bits to determine the column.
    - Outputs 4 bits.
- Total Output: 8 × 4 bits = 32 bits.

d. Permutation (P-box)

- The 32-bit output from S-boxes is permuted using a fixed P-box table.
- Output: 32 bits.

e. XOR & Swap

- The P-box output is XORed with **LPT**.
- Then, **LPT and RPT are swapped**.

After 16 rounds, there is no swap in the final round. The final output is the concatenation of RPT and LPT (not swapped).

Step 4: Final Permutation (FP)

- Final 64-bit output of 16 rounds is permuted again using a predefined table.
- Output: 64-bit **ciphertext**.

Decryption

- Same process as encryption, but subkeys are used in **reverse order** (from round 16 to 1).

Q. Why is DES considered insecure today?

DES (Data Encryption Standard) was once a widely used symmetric key encryption algorithm. However, due to advances in computational power and cryptanalysis techniques, **DES is now considered insecure** and obsolete for protecting sensitive information.

Reasons Why DES Is Insecure Today

1. Short Key Length (56 bits)

- DES uses a **56-bit effective key**, which allows only **2^{56} possible keys**.

- Modern computers can perform **brute-force attacks** (trying all key combinations) in **a matter of hours or even minutes**.
- In 1998, the **EFF (Electronic Frontier Foundation)** built a machine that cracked DES in less than 24 hours.

2. Vulnerability to Brute Force

- The **keyspace is too small** to resist exhaustive key search.
- As computing power grows, brute-force attacks are becoming faster and cheaper.

3. Availability of Better Alternatives

- **AES (Advanced Encryption Standard)** uses **128/192/256-bit keys**, and is far more secure.
- **Triple DES (3DES)** is more secure than DES but slower than AES.

Q. Explain Blowfish algorithm.

Blowfish is a **symmetric block cipher** designed by **Bruce Schneier** in 1993 as a **fast, free, and secure alternative to DES**. It is well-known for being **highly configurable**, **open-source**, and **not patented**.

Key Features of Blowfish

Attribute	**Value**
Type	Symmetric block cipher
Block Size	64 bits
Key Size	Variable: **32 to 448 bits**
Number of Rounds	16 rounds
Number of Subkeys	18 subkeys (P-array)
S-boxes	4 S-boxes, each with **256 entries** of **32-bit** values
Speed & Flexibility	Fast and free to use; good for both hardware and software implementations

---

Blowfish Algorithm Has Two Main Parts

1. **Subkey Generation**

2. **Data Encryption**

---

1. Subkey Generation

a. P-Array and S-Boxes Initialization

- **P-array:** 18 entries (P[0] to P[17]), each of **32 bits**
- **S-boxes:** 4 boxes (S1, S2, S3, S4), each has **256 entries** (32-bit each)
- These are initialized with **hexadecimal constants**

b. Key Mixing

- Input user key is divided into 32-bit segments (up to 14 keys if 448 bits)
- Each element of P-array is **XORed** with a key segment:

  P[0] = P[0] XOR K1

  P[1] = P[1] XOR K2

  ...

  P[17] = P[17] XOR K18

c. Subkey Refinement

- Encrypt a 64-bit all-zero block using current P-array and S-boxes
- Replace P[0] and P[1] with the resulting ciphertext
- Encrypt the new ciphertext → update P[2], P[3], and so on...
- Continue this for all **P-array** and **S-boxes**
- Result: All subkeys are derived from both the key and Blowfish algorithm

---

2. Encryption Process

a. Input

- **64-bit plaintext** divided into two 32-bit halves:
  - XL (Left half)
  - XR (Right half)

b. 16 Rounds

Each round does the following:

1. XL = XL XOR P[i]
2. F(XL) → result is XORed with XR

3. Swap XL and XR

Repeat for **16 rounds**

c. F Function (Key to Security)

- Input: 32-bit → divide into four 8-bit parts: XA, XB, XC, XD
- Process:

  $F(XL) = ((S1[XA] + S2[XB]) \bmod 2^{32}) \text{ XOR } S3[XC]$

  $F(XL) = (\text{Result} + S4[XD]) \bmod 2^{32}$
- Output: 32-bit

d. Post-Processing

- After 16 rounds, **swap XL and XR** one more time
- Then:

  XR = XR XOR P[17]

  XL = XL XOR P[16]
- Combine XL and XR → final **64-bit ciphertext**

Decryption

- Same as encryption, but P-array is used in **reverse order**

Visual Flow Summary

Plaintext (64 bits) → Initial Split (XL, XR)

↓

16 Rounds:

- XOR with Subkey P[i]
- F-function on XL
- XOR F(XL) with XR
- Swap halves

↓

Post-processing:

- Final XORs with P[16] and P[17]

↓

Ciphertext (64 bits)

Advantages of Blowfish

- **Fast**: Designed for high performance in software.
- **Flexible key size**: Up to 448 bits allows strong encryption.
- **Free to use**: No patents or licenses.
- **Secure**: No practical attacks have been found on full 16-round Blowfish.

Limitations of Blowfish

- **Block size is only 64 bits**: Vulnerable to **birthday attacks** on large data volumes.
- **Slow key setup**: Not suitable for applications where the key changes frequently (e.g., per message).

Modern replacement: **Twofish** (also by Schneier), and **AES** are preferred in many systems today.

Q. Explain the AES encryption process with its structure.

AES (Advanced Encryption Standard) is a **symmetric block cipher** selected by NIST in 2001 to replace DES. It is based on the **Rijndael algorithm**, developed by **Joan Daemen and Vincent Rijmen**.

AES is widely used due to its **strong security**, **speed**, and **flexibility**.

Overview

- **Type**: Symmetric key block cipher.
- **Block Size**: 128 bits (fixed).
- **Key Sizes**: 128, 192, or 256 bits (variable).
- **Rounds**:
    - **10 rounds** for 128-bit key
    - **12 rounds** for 192-bit key
    - **14 rounds** for 256-bit key
- **Standardized by**: NIST (National Institute of Standards and Technology).
- **Faster and more secure** than DES.

How AES Works

Input Format

- The 128-bit plaintext is divided into **16 bytes**.
- These 16 bytes are arranged in a **4×4 matrix**, called the **state matrix**.

Key Expansion

- The original key is expanded into **multiple round keys** using the **key expansion algorithm**.
- **Total round keys = Number of rounds + 1** (e.g., 11 keys for 128-bit AES).

Encryption Process

Steps Performed

1. **Pre-round Transformation (AddRoundKey)**
    - The initial round key (K0) is XORed with the plaintext state.
2. **Main Rounds (repeated for 9 rounds in AES-128)**

Each round consists of 4 steps:

- **SubBytes**:
    - Each byte in the matrix is replaced with a value from a predefined **S-box**.
- **ShiftRows**:
    - Each row of the matrix is shifted left by a specific number of positions:
        - Row 0: no shift
        - Row 1: 1-byte shift
        - Row 2: 2-byte shift
        - Row 3: 3-byte shift
- **MixColumns**:
    - Each column is treated as a 4-byte vector and multiplied with a fixed matrix using **Galois Field arithmetic**.
    - **Skipped in the final round**.
- **AddRoundKey**:

        - Each byte of the state is XORed with the corresponding byte of the round key.

3. **Final Round (10th round in AES-128)**
    - **SubBytes**
    - **ShiftRows**
    - **AddRoundKey**
    - (**No MixColumns** in the final round)

Decryption

The decryption process is the reverse of encryption and includes:

1. **AddRoundKey**
2. **Inverse ShiftRows**
3. **Inverse SubBytes**
4. **Inverse MixColumns** (skipped in last round)

Decryption uses the same round keys but **in reverse order**.

Advantages of AES

- **Highly secure** – resistant to all known attacks
- **Fast and efficient** – works well in hardware and software
- **Flexible** – supports 128, 192, and 256-bit keys
- **Widely adopted** – used in SSL, VPNs, file encryption, wireless security, etc.

Q. Explain Double DES

Double DES is an enhancement of the original **DES** algorithm where the plaintext is encrypted **twice** using **two different keys** (K1 and K2) to improve security.

Encryption Process in Double DES

Plaintext

↓ (Encrypt with K1)

Intermediate Ciphertext (C1)

↓ (Encrypt with K2)

Final Ciphertext (C2)

Mathematically:

C2 = E_K2(E_K1(Plaintext))

Decryption Process in Double DES

Ciphertext (C2)

↓ (Decrypt with K2)

Intermediate Ciphertext (C1)

↓ (Decrypt with K1)

Original Plaintext

Mathematically:

Plaintext = D_K1(D_K2(C2))

Why Use Double DES?

- Using two keys **K1** and **K2**, each of n bits (e.g. 56-bit DES keys), would **theoretically require 2^112 operations** to brute-force both keys.
- This is much harder than breaking regular DES (which only requires 2^56 operations).

But Double DES Is Vulnerable!

Meet-in-the-Middle Attack:

- This attack reduces the effective security of 2DES to about **2^57 operations**, which is **not much better** than single DES.
- It works by:
    1. Encrypting the plaintext with **all possible K1 values**, storing the results.
    2. Decrypting the ciphertext with **all possible K2 values**, and checking for a match in step 1.

That's why **Triple DES (3DES)** was developed as a stronger alternative.

Q. Explain Triple DES

Triple DES is a symmetric key encryption algorithm that applies **DES three times** to each data block using **three keys**: K1, K2, and K3.

Encryption Process in 3DES (with 3 keys)

Steps:

1. **Encrypt** the 64-bit plaintext block with K1:
   Step 1 → E_K1(Plaintext)
2. **Decrypt** the result using K2:
   Step 2 → D_K2(Result of Step 1)
3. **Encrypt** the result again using K3:
   Step 3 → E_K3(Result of Step 2)

Final Output:

Ciphertext = E_K3(D_K2(E_K1(Plaintext)))

This process is called **Encrypt-Decrypt-Encrypt (EDE)**.

Decryption Process

To decrypt the ciphertext, reverse the process:

1. **Decrypt** with K3
2. **Encrypt** with K2
3. **Decrypt** with K1

Formula:

Plaintext = D_K1(E_K2(D_K3(Ciphertext)))

Key Information

Feature	**Details**
Block Size	64 bits
Key Size	3 keys × 56 bits = **168 bits** total
Effective Key Strength	~112 bits (due to meet-in-the-middle resistance)
Security	Much stronger than DES or 2DES
Speed	Slower than single DES and AES

Feature	Details
Use Cases	Financial systems, legacy systems, etc.

Why Use 3DES?

- Protects against **brute-force** and **meet-in-the-middle** attacks.
- Allows backward compatibility with older DES systems.
- Was widely adopted before **AES** became the new standard.

Disadvantages

- **Slower** than AES
- **Block size is small (64 bits)**, making it vulnerable to block collisions over large data
- Considered **deprecated** in modern security standards like NIST

Q. Compare DES, Triple DES, and AES.

Feature	DES	Triple DES (3DES)	AES
Full Name	Data Encryption Standard	Triple Data Encryption Algorithm	Advanced Encryption Standard
Developed By	IBM (1970s), adopted by NIST	NIST (upgrade to DES)	Rijndael (Joan Daemen & Rijmen)
Key Size	56 bits (64 with parity)	112 or 168 bits (2 or 3 keys)	128, 192, or 256 bits
Block Size	64 bits	64 bits	128 bits
Structure	Feistel Network	Feistel Network (applies DES 3 times)	Substitution-Permutation Network
Rounds	16	48 (16 × 3)	10 (128-bit), 12 (192), 14 (256)
Security	Weak (brute-forceable)	Moderate (still used in legacy systems)	Strong (resistant to all known attacks)
Speed	Fast (but insecure)	Slow (due to 3 encryptions per block)	Fast and efficient in hardware/software

Feature	DES	Triple DES (3DES)	AES
Status	Deprecated	Deprecated (used for backward compatibility)	Current Standard

Q. Explain RSA algorithm with numerical example.

RSA is a widely used **asymmetric encryption algorithm** invented by **Rivest, Shamir, and Adleman** in 1977. It uses **two keys**:

- **Public key** for encryption
- **Private key** for decryption

The security of RSA is based on the **mathematical difficulty of factoring large prime numbers**.

Overview

- Developed by **Ron Rivest** in 1994.
- **Type**: Symmetric key block cipher.
- Known for:
  - **Simplicity**
  - **Flexibility**
  - **Efficiency**
- Performs operations on **words**, not just bytes or bits.

Key Parameters

RC5 is highly parameterized with the following:

- **w** = Word size (can be 16, 32, or 64 bits)
- **r** = Number of rounds (0 to 255)
- **b** = Key length in bytes (0 to 255)

For example, RC5-32/12/16 means:

- Word size = 32 bits
- Number of rounds = 12
- Key size = 16 bytes (128 bits)

RC5 Structure

The algorithm consists of **three components**:

1. **Key Expansion**
2. **Encryption**
3. **Decryption**

---

1. Key Expansion

- Converts the secret key into an array of **subkeys** S[].
- Number of subkeys: $2 \times (r + 1)$ → e.g., for 12 rounds, we need **26 subkeys**.
- Steps:
    1. **Initialize S[]** using predefined constants.
    2. **Mix secret key into S[]** using XOR, addition, and rotations.
    3. This step ensures strong key mixing and increases resistance to attacks.

---

2. Encryption Process

Input: 64-bit plaintext → split into two halves:

- A and B (each of **w** bits, e.g., 32 bits)

Steps:

1. **Pre-whitening**:
    - $A = A + S[0]$
    - $B = B + S[1]$
2. **Rounds** (for i = 1 to r):
    - $A = ((A \oplus B) \lll B) + S[2*i]$
    - $B = ((B \oplus A) \lll A) + S[2*i + 1]$

Here, <<< denotes **left circular shift** (rotate left).

3. Final output after all rounds: concatenation of A and B → **ciphertext**

---

3. Decryption Process

Reverses the encryption steps using the same subkeys but in reverse order:

1. For i = r downto 1:
    - B = ((B - S[2*i + 1]) >>> A) ⊕ A
    - A = ((A - S[2*i]) >>> B) ⊕ B

>>> is right circular shift.

2. **Post-processing**:
    - B = B - S[1]
    - A = A - S[0]

Key Features of RC5

Feature	Description
Block Size	2 × word size (e.g., 64 bits if w = 32)
Key Size	Up to 2040 bits (255 bytes)
Rounds	0–255 (commonly 12 or 16)
Speed	High — very fast on most hardware
Memory Usage	Low — minimal resource requirement
Security	Depends on key length and round count

Q. What is the Knapsack cryptosystem?

The **Knapsack cryptosystem** (also called the **Merkle–Hellman Knapsack cipher**) is one of the earliest **public-key cryptographic algorithms**, invented by **Ralph Merkle and Martin Hellman** in 1978.

It is based on the mathematical problem known as the **Subset Sum Problem**, a type of **NP-complete problem**, which is easy to verify but hard to solve—making it suitable for cryptography.

The **Knapsack Problem** is a classic optimization problem:

Given a set of items, each with a weight and value, select a subset of items such that the total weight is within a limit and the total value is maximized.

In Cryptography:

It's turned into a **subset sum problem**:

- Given a set of numbers and a sum, determine which numbers add up to the sum.

Knapsack Cryptosystem Overview

Component	Type
Private Key	Easy (super-increasing) knapsack
Public Key	Hard knapsack (modular transformation of private key)
Encryption	Done using **public key**
Decryption	Done using **private key**

Steps to Implement Knapsack Encryption

1. Create a Super-Increasing Sequence (Private Key)

A **super-increasing sequence** is a sequence where each element is **greater than the sum of all previous elements**.

Example:

Private Key (super-increasing): {1, 2, 4, 10, 20, 40}

Valid because:
1+2 < 4,
1+2+4 < 10, etc.

---

2. Generate Public Key

Choose:

- A **modulus m** > sum of private key (e.g. m = 110)
- A **multiplier n** that is **coprime with m** (e.g. n = 31)

Public Key Calculation:
Each public key element = (private key element × n) mod m

Private	×31	mod 110	→ Public
1	31	31	31
2	62	62	62
4	124	14	14
10	310	90	90

Private	×31	mod 110	→ Public
20	620	70	70
40	1240	30	30

Public Key = {31, 62, 14, 90, 70, 30}

Encryption Process

1. **Convert plaintext to binary**, split into 6-bit chunks (since public key has 6 elements).
   Example:
   Binary Plaintext = 100100 111100 101110
2. For each 6-bit block, **multiply bit-by-bit with public key** and sum the values.

Example:

- Block: 100100
  → 1×31 + 0×62 + 0×14 + 1×90 + 0×70 + 0×30 = **121**
- Block: 111100
  → 1×31 + 1×62 + 1×14 + 1×90 + 0×70 + 0×30 = **197**
- Block: 101110
  → 1×31 + 0×62 + 1×14 + 1×90 + 1×70 + 0×30 = **205**

Ciphertext = {121, 197, 205}

Decryption Process (with Private Key)

1. Compute the **modular inverse** of n mod m, say n^{-1}.
2. Multiply each ciphertext value by n^{-1} mod m.
3. Use the **super-increasing sequence** to figure out which elements sum up to the result (this is easy due to the super-increasing property).
4. Convert selected bits back to binary → reconstruct plaintext.

Q. Explain Diffie-Hellman key exchange with steps and example.

The **Diffie–Hellman Key Exchange** is a method for **two parties to securely generate a shared secret key** over an **insecure public channel** without transmitting the key itself.

Invented by **Whitfield Diffie and Martin Hellman** in 1976, it is the foundation of **public-key cryptography**.

Basic Idea

- Based on the **difficulty of computing discrete logarithms** in modular arithmetic.
- Both parties use agreed-upon public values but keep **private secrets**.
- They compute the **same shared key independently** using those secrets.

Notation

Symbol	**Meaning**
p	A large **prime number** (public)
g	A primitive root modulo p (generator) (public)
a	Alice's private key (secret)
b	Bob's private key (secret)
A = g^a mod p	Alice's public value (sent to Bob)
B = g^b mod p	Bob's public value (sent to Alice)
K	Shared secret key = B^a mod p = A^b mod p

Steps in Diffie–Hellman Key Exchange

Step 1: Agree on Public Parameters

Let:

- p = 23 (a prime number)
- g = 5 (a primitive root modulo 23)

These values are **public**.

Step 2: Choose Private Keys

- Alice chooses **private key** a = 6
- Bob chooses **private key** b = 15

These values are **secret**.

Step 3: Compute Public Values

- Alice computes A = g^a mod p = 5^6 mod 23 = 15625 mod 23 = **8**
- Bob computes B = g^b mod p = 5^15 mod 23 = 30517578125 mod 23 = **2**

Alice sends **A = 8** to Bob,
Bob sends **B = 2** to Alice.

Step 4: Compute Shared Secret Key

- Alice computes: K = B^a mod p = 2^6 mod 23 = 64 mod 23 = **18**
- Bob computes: K = A^b mod p = 8^15 mod 23 = 18

Both get the same shared secret key: K = 18

Security of Diffie–Hellman

- An eavesdropper sees p, g, A, and B but **cannot compute the key K** without solving:

  gamod p=A⇒Find a(Discrete Log Problem)

- This is computationally hard for large p (2048-bit or more in real systems).

Applications

- Secure key exchange in **TLS/SSL**, **VPNs**, **Wi-Fi (WPA3)**, etc.
- Used to generate symmetric keys for **AES**, **3DES**, etc.

Q. Explain the ElGamal encryption algorithm.

The **ElGamal encryption algorithm** is a **public-key cryptographic system** based on the **Diffie–Hellman key exchange**. It provides both **confidentiality** and **semantic security**, meaning the same message will encrypt to **different ciphertexts every time** due to its use of randomness.

It was proposed by **Taher ElGamal** in 1985.

Overview

Feature	Description
Type	Asymmetric key cryptography
Key Type	Public & Private key pair
Used For	Secure message encryption
Based On	Discrete logarithm problem

Key Generation (by Sam)

1. **Choose a large prime number q**
2. **Choose a generator g** of a **cyclic group** modulo q
   - This means any number in the group can be written as g^x mod q
3. **Select a private key a**
   - A random number such that $1 < a < q$
4. **Compute the public value h = g^a mod q**

Public Key: (q, g, h)

Private Key: a

Encryption (by Rita)

Rita wants to send message **M** to Sam using his **public key**.

1. **Choose a random k**, where gcd(k, q) = 1
2. **Compute:**
   - p = g^k mod q
   - s = h^k mod q = (g^a)^k = g^(ak)
3. **Encrypt message:**
   - Ciphertext = (p, M × s mod q)
4. **Send (p, c) to Sam**

Decryption (by Sam)

1. **Compute shared secret s again:**
   - s = p^a mod q = (g^k)^a = g^(ak)
2. **Recover message M:**
   - M = c × s^{-1} mod q (use modular inverse of s)

Summary Table

Step	Description
Key Generation	Sam creates public & private keys
Encryption	Rita uses Sam's public key to encrypt message

Step	Description
Decryption	Sam uses private key to recover message

Example (Simplified for clarity)

1. Sam chooses:
   - $q = 17, g = 3, a = 15$
   - Compute $h = g^a \bmod q = 3^{15} \bmod 17 = 6$
   - Public Key = (17, 3, 6)
2. Rita wants to send M = 13
   - Chooses k = 7
   - Computes:
     - $p = 3^7 \bmod 17 = 11$
     - $s = 6^7 \bmod 17 = 7$
     - Ciphertext = $c = (13 \times 7) \bmod 17 = 6$
     - Sends (p = 11, c = 6)
3. Sam decrypts:
   - Computes $s = 11^{15} \bmod 17 = 7$
   - $s^{-1} \bmod 17 = 5$
   - $M = 6 \times 5 \bmod 17 = 13$

Recovered message: 13

Q. How is ElGamal encryption different from RSA?

Aspect	ElGamal	RSA
Inventor	Taher ElGamal (1985)	Rivest, Shamir, Adleman (1977)
Based on	Discrete Logarithm Problem	Integer Factorization Problem
Randomness	**Probabilistic** – uses a fresh random number k every time	**Deterministic** (unless padding like OAEP is used)

Aspect	ElGamal	RSA
Encryption Formula	C1 = g^k mod p, C2 = M × y^k mod p	C = M^e mod n
Ciphertext	Two values: (C1, C2)	One value: C
Plaintext Recovery	Requires modular inverse of shared secret	Uses private exponent d: M = C^d mod n
Message Size Limit	Message must be < p (a prime modulus)	Can encrypt any number < n
Security Basis	Hardness of computing discrete logs in mod p	Hardness of factoring large integers
Semantic Security	**Yes**, even without padding (due to random k)	**No**, unless padding is added (e.g., OAEP)
Use Cases	Digital signatures, secure messaging (used in PGP, GnuPG)	Email security, digital signatures, certificates (widely used in TLS)

Q. Compare public key and private key cryptography.

Aspect	Private Key Cryptography	Public Key Cryptography
Also Called	Symmetric Cryptography	Asymmetric Cryptography
Number of Keys	One key (same for encryption & decryption)	Two keys: Public key and Private key
Key Sharing	Must be shared securely	Public key is openly shared; private key is kept secret
Speed	**Faster**, efficient for large data	**Slower**, due to complex math
Security Depends On	Secrecy of the single shared key	Secrecy of the private key
Key Management	Hard in large systems (n users = n^2 keys)	Easier (public key infrastructure)
Encryption Example	AES, DES, Blowfish	RSA, ElGamal, ECC
Used For	Encrypting large files, VPNs, SSL data transfer	Key exchange, digital signatures, email security

Aspect	Private Key Cryptography	Public Key Cryptography
Vulnerability	Key exposure compromises both sides	Only private key holder can decrypt or sign