Q) Problems in Public Key and Private Key Cryptography

1. Problems in Private Key (Symmetric) Cryptography
   (1) Key Distribution Problem
   (2) Key Management Problem
   (3) Scalability Issues
   (4) Lack of Non-Repudiation: Since both parties use the same key, it is difficult to prove who sent the message., No digital signature functionality.

2. Problems in Public Key (Asymmetric) Cryptography
   (1) Slow Performance
   (2) Key Authenticity Problem
   (3) Large Key Size
   (4) Security Based on Mathematical Assumptions
   (5) Complex Key Management Infrastructure

Q) Perfect Secrecy in cryptography

Perfect secrecy is a property of an encryption system in which the ciphertext reveals absolutely no information about the plaintext.

Even if an attacker has unlimited computational power and full knowledge of the encryption algorithm, they cannot gain any information about the original message from the ciphertext.

The concept of perfect secrecy was formally defined by **Claude Shannon**, the father of information theory.

Formal Definition

An encryption scheme has perfect secrecy if:

P(M|C)=P(M)P(M \mid C) = P(M)P(M|C)=P(M)

This means that the probability of a message MMM given the ciphertext CCC is equal to the prior probability of MMM.

In simple terms, observing the ciphertext does not change the attacker's knowledge about the plaintext.

Intuition Behind Perfect Secrecy

- Before seeing the ciphertext, every plaintext has some probability.
- After seeing the ciphertext, the probabilities remain exactly the same.
- Thus, the ciphertext provides zero additional information.

One-Time Pad (OTP) – Example of Perfect Secrecy

$$\text{Ciphertext} = \text{Plaintext} \oplus \text{Key}$$

Since the key is completely random, every possible plaintext is equally likely for a given ciphertext.

Conditions Required for Perfect Secrecy

For an encryption scheme to have perfect secrecy:

1. The key must be at least as long as the message.
2. The key must be truly random.
3. The key must never be reused.
4. The key must remain secret.

If any of these conditions are violated, perfect secrecy is lost.

Limitations of Perfect Secrecy

Although theoretically secure, perfect secrecy is impractical in most real-world systems because:

- Very large key storage is required
- Secure key distribution is difficult
- Key management becomes complex

Therefore, modern encryption systems (like AES) aim for **computational security**, not perfect secrecy.

Q) Forward Secrecy and Backward Secrecy

In secure communication systems, protecting past and future session keys from compromise is very important.

Two important security properties related to key management are:

- **Forward Secrecy (FS)**

- **Backward Secrecy (BS)**

These properties ensure that compromise of one key does not affect other session keys.

---

1. Forward Secrecy (Perfect Forward Secrecy – PFS)

Definition

Forward Secrecy means that compromise of a long-term private key does **not** compromise past session keys.

In simple terms:
Even if an attacker obtains the server's private key today, they cannot decrypt previously recorded encrypted sessions.

Why Is It Needed?

Attackers often:

- Record encrypted traffic.
- Wait until they obtain the private key.
- Attempt to decrypt past communications.

Forward secrecy prevents this.

How It Is Achieved

Forward secrecy is typically achieved using **ephemeral key exchange mechanisms**, such as:

- **Diffie–Hellman key exchange**
- Ephemeral Diffie–Hellman (DHE)
- Elliptic Curve Diffie–Hellman Ephemeral (ECDHE)

Each session:

- Generates a temporary (ephemeral) key.
- Discards it after the session ends.
- Keys are not stored long-term.

Thus, even if long-term keys are leaked, past session keys remain secure.

Example

Modern **Transport Layer Security** implementations use ECDHE to provide forward secrecy.

---

2. Backward Secrecy

Definition

Backward Secrecy ensures that compromise of a current session key does not allow an attacker to decrypt future communications.

In other words:
If today's session key is exposed, it should not affect future sessions.

Why Is It Needed?

Without backward secrecy:

- If one session key leaks,
- Attacker can predict future keys,
- Future communication becomes insecure.

How It Is Achieved

Backward secrecy is achieved by:

- Generating independent session keys for each session.
- Using fresh random values.
- Avoiding reuse of old keys.
- Secure key update mechanisms in group communication systems.

Feature	**Forward Secrecy**	**Backward Secrecy**
Protects	Past sessions	Future sessions
If Current Key is Compromised	Past safe	Future safe
Requires	Ephemeral key exchange	Independent fresh keys
Example	DHE, ECDHE in TLS	Secure re-keying protocols

Q) Applications

1. Communication over insecure channel

2. Secure storage on insecure media

3. Authentication

How Authentication Protocols Prevent Man-in-the-Middle (MITM) Attacks

A **Man-in-the-Middle (MITM) attack** occurs when an attacker secretly intercepts and possibly modifies communication between two legitimate parties who believe they are directly communicating with each other.

In such attacks, the attacker can:

- Eavesdrop on communication
- Modify messages
- Impersonate either party

Authentication protocols are designed to prevent such attacks by verifying the identity of communicating parties before exchanging sensitive data.

What is an Authentication Protocol?

An authentication protocol is a set of rules used to verify the identity of users or systems before secure communication is established.

It ensures:

- The sender is genuine
- The receiver is genuine
- Messages are not modified

How Authentication Protocols Prevent MITM

Authentication protocols prevent MITM using the following mechanisms:

---

a. Digital Certificates

In protocols like **Transport Layer Security**, servers present a digital certificate signed by a trusted Certificate Authority (CA).

Prevention Mechanism:

- Client verifies certificate.
- Ensures the public key actually belongs to the intended server.
- Prevents attacker from substituting their own key.

---

b. Digital Signatures

Systems like **RSA** allow messages to be digitally signed.

Prevention Mechanism:

- Message is signed with private key.
- Receiver verifies using public key.
- Attacker cannot forge signature without private key.

Thus, identity and message integrity are confirmed.

---

c. Mutual Authentication

Both parties authenticate each other (not just server).

Example:

- Client verifies server.
- Server verifies client credentials.

This prevents impersonation on either side.

---

d. Challenge-Response Mechanism

- Server sends a random number (challenge).
- Client encrypts or signs it and sends back.
- Proves possession of secret key without revealing it.

Since attacker does not know the secret key, they cannot generate correct response.

---

e. Session Key Establishment

Modern protocols use:

- Public key cryptography to authenticate
- Symmetric key (like **Advanced Encryption Standard**) for secure session

This prevents attackers from injecting their own session keys.

Mechanism	How It Prevents MITM
Digital Certificates	Validates identity
Digital Signatures	Prevents message forgery
Challenge-Response	Proves possession of secret
Mutual Authentication	Prevents impersonation
Secure Key Exchange	Prevents key substitution

4. Integrity Check:

Role of CRC, Checksum, Error Detection and Correction in Integrity Checks

Introduction

Data integrity means ensuring that data remains accurate, consistent, and unaltered during transmission or storage.

During communication over a network, data may get corrupted due to:

- Noise
- Signal distortion
- Hardware faults
- Transmission errors

To detect and sometimes correct such errors, techniques like **Checksum**, **CRC (Cyclic Redundancy Check)**, and **Error Detection and Correction codes** are used.

---

1. Checksum in Integrity Verification

Definition

A checksum is a small numerical value calculated from data and transmitted along with it. The receiver recalculates the checksum to verify data integrity.

Working

1. Sender divides data into equal-sized blocks.
2. Blocks are added using binary addition.
3. The final sum (or its complement) is sent along with data.
4. Receiver performs the same calculation.
5. If results match → data is assumed correct.
6. If mismatch → error detected.

Example Use

- Internet Protocol (IP)
- TCP/UDP headers

Checksum helps detect accidental data corruption.

---

2. CRC (Cyclic Redundancy Check)

Definition

CRC is a more powerful error detection technique based on polynomial division in binary arithmetic

1. Data is treated as a binary polynomial.
2. It is divided by a predefined generator polynomial.
3. The remainder is called the CRC code.
4. Sender appends CRC to message.
5. Receiver performs division again.
6. If remainder is zero → no error detected.

Advantages

- Detects burst errors effectively
- More reliable than simple checksum
- Used in Ethernet, Wi-Fi, and storage devices

---

3. Error Detection

Error detection techniques determine whether data has been corrupted.

Common methods:

- Parity bits
- Checksum
- CRC

If error is detected:

- Data is discarded
- Request for retransmission is sent

Thus, integrity is maintained.

---

4. Error Correction

Unlike detection, error correction can automatically correct errors without retransmission.

Examples:

- Hamming Code
- Reed-Solomon Codes

These techniques:

- Add redundant bits
- Identify error location
- Correct corrupted bits

Used in:

- CDs/DVDs
- Satellite communication
- Mobile networks

Q) How Message Authentication Code (MAC) Provides Integrity and Other Security Properties

Introduction

A **Message Authentication Code (MAC)** is a cryptographic technique used to ensure **data integrity** and **authentication** of a message.

It is generated using:

- A secret key shared between sender and receiver
- A cryptographic algorithm

Only parties who possess the secret key can generate or verify the MAC.

What is a MAC?

A MAC is a fixed-size output produced by applying a MAC algorithm to a message and a secret key.

MAC=f(Key,Message)MAC = f(Key, Message)MAC=f(Key,Message)

The sender sends:

- Message
- MAC value

The receiver recomputes the MAC using the same secret key and compares the values.

If MACs match → message is authentic and unmodified.
If MACs differ → message has been tampered with.

How MAC Provides Integrity

1. Detection of Message Modification

If an attacker modifies even a single bit of the message:

- The recomputed MAC will be completely different.
- Verification fails.

Modern MAC constructions like **HMAC** are highly sensitive to changes due to hash properties.

Thus, MAC ensures:

- Data has not been altered
- Protection against accidental or malicious modification

How MAC Provides Authentication

Since MAC requires a **secret key**, only legitimate sender and receiver can compute the correct MAC.

This ensures:

- The message came from someone who knows the secret key.
- Attackers cannot forge valid MAC without key knowledge.

Common hash functions used in MAC construction include **SHA-256**.

What MAC Does NOT Provide

1. Confidentiality

MAC does not encrypt data.
The message is still visible unless encryption is applied separately.

2. Non-Repudiation

Since both sender and receiver share the same secret key:

- Either party could generate the MAC.
- Therefore, sender cannot be uniquely proven in court.

For non-repudiation, digital signatures using asymmetric cryptography (e.g., **RSA**) are required.

Working of MAC in Communication

3. Sender computes MAC using secret key.
4. Sends message + MAC.
5. Receiver computes MAC again.
6. Compares values.
7. If equal → integrity and authenticity verified.

Q) Security Basis of RSA and Diffie–Hellman

Modern public key cryptography is based on certain **mathematical problems that are computationally hard to solve**.

Two important cryptosystems are:

- **RSA**
- **Diffie–Hellman Key Exchange**

Their security depends on different hard mathematical problems.

---

1. RSA and the Integer Factorization Problem

Basic Idea

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

Working Principle

1. Choose two large prime numbers ppp and qqq.
2. Compute:

   n=p×qn = p \times qn=p×q

3. Public key contains nnn.
4. Private key depends on knowing ppp and qqq.

To break RSA, an attacker must:

- Factor nnn back into ppp and qqq.

Why Is It Hard?

- For small numbers, factorization is easy.
- For very large numbers (2048-bit or more), factorization becomes computationally infeasible using classical computers.
- No known efficient algorithm for large integer factorization (classically).

Thus, RSA's security relies on the **Integer Factorization Problem**.

---

2. Diffie–Hellman and the Discrete Logarithm Problem

Basic Idea

The security of **Whitfield Diffie** and **Martin Hellman**'s Diffie–Hellman scheme relies on the **Discrete Logarithm Problem (DLP)**.

Working Principle

Let:

- ggg = generator
- ppp = large prime number

Two users compute values like:

gamod pg^a \mod pgamodp gbmod pg^b \mod pgbmodp

The shared secret becomes:

gabmod pg^{ab} \mod pgabmodp

Where Is the Hard Problem?

Even if attacker knows:

- ggg

- ppp
- gamod pg^a \mod pgamodp

They must find:

aaa

This requires solving the **discrete logarithm problem**, which is computationally hard for large primes.

Comparison of Hard Problems

Cryptosystem	Hard Problem	Why Hard?
RSA	Integer Factorization	Factoring large composite numbers
Diffie–Hellman	Discrete Logarithm Problem	Finding exponent from modular exponentiation

---

Important Note (Exam Enhancement Point)

Both problems are considered hard on **classical computers**, but quantum algorithms (like Shor's Algorithm) can solve:

- Integer factorization
- Discrete logarithm problem

This threatens both RSA and Diffie–Hellman in the quantum computing era.