MODULE - 3 SYLLABUS:

PUBLIC KEY CRYPTOGRAPHY AND CRYPTOGRAPHIC DATA INTEGRITY ALGORITHMS

Public Key Cryptography: RSA, Diffie-Hellman key exchange, Elgamal cryptographic system.

Cryptographic Data Integrity Algorithms: Hash Functions - Simple hash functions, Secure Hash Algorithm SHA-512; Message Authentication Codes – Requirements, Functions, Security of MACs, HMAC; Digital signatures - Schnorr Digital signature scheme;

1. Public Key Cryptography

1.1 RSA Algorithm

Description:
RSA (Rivest-Shamir-Adleman) is one of the first public-key cryptosystems and is widely used for secure data transmission. It relies on the difficulty of factoring large composite numbers.

Algorithm Steps:

1. **Key Generation**:

- Choose two distinct large prime numbers p and q.
- Compute $n = p \times q$ (modulus).
- Compute $\phi(n) = (p-1)(q-1)$ (Euler's totient function).
- Choose a public exponent e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$.
- Compute the private key d such that $d \cdot e \equiv 1 \pmod{\phi(n)}$.
- Public key: (e, n); Private key: d.

2. **Encryption:**

- Given a plaintext message M, convert it to an integer m such that $0 \leq m < n$.
- Compute ciphertext $c \equiv m^e \pmod{n}$.

3. **Decryption:**

- Compute plaintext $m \equiv c^d \pmod{n}$.
- Convert m back to the original message M.

Example:

- Select $p = 61$, $q = 53$.
- Compute $n = p \times q = 61 \times 53 = 3233$.
- $\phi(n) = (p-1)(q-1) = 60 \times 52 = 3120$.
- Choose $e = 17$ ($\gcd(17, 3120) = 1$).
- Compute d: $d \cdot 17 \equiv 1 \pmod{3120}$, $d = 2753$.
- Public key: $(17, 3233)$; Private key: 2753.

Encryption of $m = 65$:
$c = 65^{17} \mod 3233 = 2790$.

Decryption of $c = 2790$:
$m = 2790^{2753} \mod 3233 = 65$.

1.2 Diffie-Hellman Key Exchange

Description:

Diffie-Hellman is a method for two parties to securely establish a shared secret over an insecure channel**.**

Algorithm Steps:

1. Select a large prime number p and a primitive root modulo p, g.
2. **Key Exchange**:
   - Alice selects a private key a and computes $A = g^a \mod p$.
   - Bob selects a private key b and computes $B = g^b \mod p$.
   - Alice and Bob exchange A and B over the network.
3. **Shared Secret Computation**:
   - Alice computes $S = B^a \mod p$.
   - Bob computes $S = A^b \mod p$.
   - Both derive the same shared secret S.

Example:

- $p = 23$, $g = 5$.
- Alice chooses $a = 6$: $A = 5^6 \mod 23 = 8$.
- Bob chooses $b = 15$: $B = 5^{15} \mod 23 = 19$.
- Exchange: Alice sends $A = 8$, Bob sends $B = 19$.
- Shared secret: $S = 19^6 \mod 23 = 2$ (Alice); $S = 8^{15} \mod 23 = 2$ (Bob).

1.3 ElGamal Cryptographic System

Description:

The ElGamal cryptographic system is a public-key cryptosystem based on the discrete logarithm problem. It is used for encryption and digital signatures, providing strong security due to the computational difficulty of solving discrete logarithms.

Algorithm Steps:

1. **Key Generation:**
   - Choose a large prime number p and a primitive root modulo p, g.
   - Select a private key x such that $1 \leq x \leq p-2$.
   - Compute the public key $y = g^x \mod p$.
   - Public key: (p, g, y); Private key: x.
2. **Encryption:**
   - To encrypt a message m, choose a random integer k such that $1 \leq k \leq p-2$.
   - Compute $c_1 = g^k \mod p$ and $c_2 = m \cdot y^k \mod p$.
   - Ciphertext: (c_1, c_2).
3. **Decryption:**
   - Given ciphertext (c_1, c_2), compute $s = c_1^x \mod p$.
   - Recover the plaintext m as $m = c_2 \cdot s^{-1} \mod p$, where s^{-1} is the modular multiplicative inverse of s modulo p.

Example:

- $p = 17, g = 3$.
- Private key $x = 15$, compute $y = 3^{15} \mod 17 = 6$.
- Public key: $(17, 3, 6)$.
- Encrypt $m = 13$:
  - Choose $k = 10$: $c_1 = 3^{10} \mod 17 = 5$.
  - Compute $c_2 = 13 \cdot 6^{10} \mod 17 = 13 \cdot 15 \mod 17 = 8$.
  - Ciphertext: $(5, 8)$.
- Decrypt $(5, 8)$:
  - Compute $s = 5^{15} \mod 17 = 15$.
  - Compute $s^{-1} \mod 17 = 13$.
  - Recover $m = 8 \cdot 13 \mod 17 = 13$.

Applications:

- Secure email communication.
- Digital signatures in cryptocurrency systems like Ethereum.

2. Cryptographic Data Integrity Algorithms

Cryptographic data integrity algorithms are designed to protect data from unauthorized alterations. These algorithms verify the integrity and authenticity of data by generating a "fingerprint" or "digest" that is unique to the original message or file. Below, we will explore **Hash Functions**, **Message Authentication Codes (MACs)**, and **Digital Signatures**, which are crucial components for data integrity.

2.1 Hash Functions

2.1.1 Definition and Purpose

A **hash function** takes an input (message) of arbitrary length and produces a fixed-size output (hash or digest). It is used to verify data integrity, create digital signatures, and store passwords securely.

Key characteristics of hash functions:

1. **Deterministic**: For the same input, the output hash is always the same.
2. **Fixed-size Output**: No matter the size of the input, the output is always of a fixed size (e.g., SHA-512 generates a 512-bit output).
3. **Pre-image Resistance**: It is computationally infeasible to reverse-engineer the original input from the hash.
4. **Collision Resistance**: It is computationally infeasible to find two distinct inputs that produce the same hash.
5. **Avalanche Effect**: A small change in the input drastically changes the hash.

2.1.2 Simple Hash Functions

Simple hash functions are basic algorithms that do not provide the security necessary for cryptographic purposes. They can be used for tasks like checksums or simple data verifications, but they are not designed to withstand cryptographic attacks.

Example: Simple Checksum

A simple checksum can be computed by summing the ASCII values of the characters in a string.

For the input **"HELLO"**:

1. Convert characters to ASCII values:
    - H = 72

- E = 69
- L = 76
- L = 76
- O = 79

2. Add them up: 72+69+76+76+79=37272 + 69 + 76 + 76 + 79 = 37272+69+76+76+79=372

Thus, the checksum for "HELLO" is 372.

Limitations of Simple Hash Functions

- **Collision Vulnerability**: Multiple inputs may lead to the same checksum.
- **Lack of Security**: Simple functions don't provide pre-image or collision resistance, making them unsuitable for cryptographic applications.

2.1.3 Secure Hash Algorithms (SHA-512)

SHA-512 is part of the SHA-2 family of hash functions, developed by the NSA. It is a widely used cryptographic hash function that outputs a 512-bit hash value. SHA-512 is considered secure and is used in blockchain, file integrity verification, digital signatures, and more.

SHA-512 Algorithm Overview:

SHA-512 works by taking an input message, padding it, and then processing it in blocks of 1024 bits. The algorithm employs bitwise operations, modular arithmetic, and predefined constants to ensure the hash is unique and resistant to collisions.

1. **Padding**: The message is padded so its length is a multiple of 1024 bits. A '1' bit is added, followed by zeroes, and the length of the original message is appended at the end.
2. **Message Processing**: The padded message is divided into 1024-bit blocks. Each block undergoes 80 rounds of processing where each round involves modular addition, XOR operations, and word expansions.
3. **Final Hash**: After processing all blocks, the final 512-bit hash value is produced.

2.2 Message Authentication Codes (MACs)

2.2.1 Definition and Purpose

A **Message Authentication Code (MAC)** is a cryptographic code used to authenticate a message. It ensures that the message has not been tampered with and verifies the identity of the sender. A MAC is generated using both the message and a secret key.

MACs provide two main features:

- **Integrity**: Ensures the message has not been altered.
- **Authentication**: Verifies the identity of the sender.

2.2.2 Requirements for MACs

1. **Confidentiality**: Only parties with the shared secret key can generate or verify the MAC.
2. **Uniqueness**: Each MAC must be unique for different messages, even with the same key.
3. **Tamper Resistance**: Any change to the message will invalidate the MAC.

2.2.3 HMAC (Hash-based Message Authentication Code)

HMAC combines a hash function (like SHA-256 or SHA-512) and a secret key to produce a secure MAC. The strength of HMAC depends on the underlying hash function's security.

HMAC Algorithm Steps:

1. **Key Preparation**: If the key is longer than the hash block size, it is hashed first. If it is shorter, it is padded with zeros.
2. **Inner Hash Calculation**: The key is XORed with the inner padding (IPAD), and then the message is concatenated to it. This combination is then hashed.
3. **Outer Hash Calculation**: The key is XORed with the outer padding (OPAD), and the inner hash is appended. This combination is then hashed.
4. **Final MAC Output**: The resulting value from the outer hash is the HMAC.

HMAC Formula:

$$HMAC(K, M) = H((K \oplus OPAD) \| H((K \oplus IPAD) \| M))$$

2.3 Digital Signatures

2.3.1 Definition and Purpose

A **digital signature** is a cryptographic scheme used to authenticate the identity of a sender and verify the integrity of a message. It ensures that the message has not been altered after it was signed, and it prevents the sender from denying authorship (non-repudiation).

A digital signature typically involves two steps:

1. **Signing**: The sender signs the message (or its hash) with their private key.
2. **Verification**: The recipient verifies the signature using the sender's public key.

2.3.2 Schnorr Digital Signature Scheme

The **Schnorr signature scheme** is based on the difficulty of solving the discrete logarithm problem. It is one of the most efficient and secure signature schemes.

Schnorr Signature Algorithm:

1. **Key Generation**:
    - Choose a large prime p and a generator g of the group $\mathbb{Z}_p$.
    - Select a private key x, where x is a randomly chosen integer.
    - Compute the public key $y = g^x \mod p$.
2. **Signing**:
    - Choose a random number k and compute $r = g^k \mod p$.
    - Compute the hash $e = H(m \| r)$, where m is the message.
    - Compute the signature $s = k - x \cdot e \mod (p-1)$.
    - The signature is (r, s).
3. **Verification**:
    - Compute $v = g^s \cdot y^e \mod p$.
    - If $v = r$, the signature is valid.

Example:

1. **Key Generation**:
    - Let $p = 23$, $g = 5$, and $x = 6$.
    - Public key $y = g^x \mod p = 5^6 \mod 23 = 8$.

2. **Signing**:

   - Message: "HELLO"
   - Random number $k = 15$.
   - Compute $r = g^k \mod p = 5^{15} \mod 23 = 19$.
   - Compute the hash $e = H(\text{"}HELLO\text{"} \| 19) = H(\text{"}HELLO19\text{"})$.
   - Compute $s = k - x \cdot e \mod (p-1)$.
   - Signature: (r, s).

3. **Verification**:

   - Compute $v = g^s \cdot y^e \mod p$.
   - If $v = r$, the signature is valid.