**UNIT II:**

**Public key Cryptogaphy Principles, RSA algoritbm, Key Management, Diffie-Helknan Key Exchange, Elliptic Curve Cryptography. Message authentication and Hash Functions, Authentication Requirements and Functions, Message Authentication, Hash Functions and MACs Hash and MAC Algorithms SHA-512, HMAC.**

## Public key Cryptography & Principles:

Public key cryptography, also known as asymmetric cryptography, is a system that uses pairs of keys for secure communication. One key is public and can be shared openly, while the other is private and must be kept secret. Here are the main principles:

1. Key Pair:
   - Public Key: This key is shared with everyone. It's used to encrypt messages or verify signatures.
   - Private Key: This key is kept secret by the owner. It's used to decrypt messages or sign data.
2. Encryption and Decryption:
   - A message encrypted with a public key can only be decrypted with the corresponding private key.
   - This ensures that only the intended recipient (who owns the private key) can read the message.
3. Digital Signatures:
   - A private key can be used to sign data or a message.
   - Anyone with the corresponding public key can verify the signature, ensuring that the message was not tampered with and confirming the identity of the sender.
4. Confidentiality:
   - Public key cryptography ensures confidentiality by encrypting the message with the recipient's public key. Only the recipient with the private key can decrypt it.
5. Authentication:
   - Public key cryptography can also be used for authentication. A sender signs a message with their private key, and the receiver verifies it using the sender's public key.
6. Non-repudiation:
   - If a message is signed with a private key, the sender cannot later deny having sent the message because only they have access to the private key.
7. Key Distribution:

- Unlike symmetric cryptography, public key cryptography avoids the problem of securely sharing a secret key, as the public key can be openly distributed without compromising security.

8. Mathematical Foundations:

- Public key cryptography relies on complex mathematical problems, such as factoring large prime numbers (RSA) or solving discrete logarithms (Elliptic Curve Cryptography), which are computationally difficult to reverse.

9. Common Algorithms:

- RSA: Based on the difficulty of factoring large numbers.
- Elliptic Curve Cryptography (ECC): Based on the mathematics of elliptic curves, offering similar security with smaller key sizes.
- Diffie-Hellman: A method for exchanging cryptographic keys over a public channel.

Public key cryptography is widely used in secure communication protocols like SSL/TLS (for secure web browsing), email encryption (PGP), and digital certificates (PKI).

RSA Cryptosystem This cryptosystem is one the initial system. It remains most employed cryptosystem even today. The system was invented by three scholars Ron Rivest, Adi Shamir, and Len Adleman and hence, it is termed as RSA cryptosystem.

We will see two aspects of the RSA cryptosystem, firstly generation of key pair and secondly encryption-decryption algorithms.

**Generation of RSA Key Pair**: Each person or a party who desires to participate in communication using encryption needs to generate a pair of keys, namely public key and private key. The process followed in the generation of keys is described below.

**Generate the RSA modulus (n)**

Select two large primes, p and q.

Calculate n=p*q.

For strong unbreakable encryption, let n be a large number, typically a minimum of 512 bits.

**Find Derived Number (e):**

Number e must be greater than 1 and less than $(p - 1)(q - 1)$.

There must be no common factor for e and $(p - 1)(q - 1)$ except for 1.

In other words two numbers e and $(p – 1)(q – 1)$ are coprime.

**Form the public key:**

The pair of numbers (n, e) form the RSA public key and is made public.

Interestingly, though n is part of the public key, difficulty in factorizing a large prime number ensures that attacker cannot find in finite time the two primes (p & q) used to obtain n.

This is strength of RSA

**Generate the private key:**

Private Key d is calculated from p, q, and e. For given n and e, there is unique number d.

Number d is the inverse of e modulo (p - 1)(q – 1). This means that d is the number less than (p - 1)(q - 1) such that when multiplied by e, it is equal to 1 modulo (p - 1)(q - 1). This relationship is written mathematically as follows

$$ed = 1 \bmod (p - 1)(q - 1)$$

The Extended Euclidean Algorithm takes p, q, and e as input and gives d as output.

**Example An example of generating RSA Key pair is given below.**

(For ease of understanding, the primes p & q taken here are small values.

Practically, these values are very high).

- Let two primes be p = 7 and q = 13.

Thus, modulus n = pq = 7 x 13 = 91.

- Select e = 5, which is a valid choice since there is no number that is common factor of 5 and $(p - 1)(q - 1) = 6 \times 12 = 72$, except for 1.

- The pair of numbers (n, e) = (91, 5) forms the public key and can be made available to anyone whom we wish to be able to send us encrypted messages.

- Input p = 7, q = 13, and e = 5 to the Extended Euclidean Algorithm. The output will be d = 29

. - Check that the d calculated is correct by computing $- de = 29 \times 5 = 145 = 1 \bmod 72$

- Hence, public key is (91, 5) and private keys is (91, 29).

**The Diffie-Hellman Key Exchange (DHKE):**

It is a cryptographic protocol that allows two parties to securely share a secret key over an insecure communication channel without ever transmitting the key itself. It was one of the first practical implementations of public key exchange and serves as the foundation for many encryption protocols.

**Key Principles of Diffie-Hellman Key Exchange:**

1. **Mathematical Basis**:

   The Diffie-Hellman key exchange is based on the difficulty of solving the **discrete logarithm problem**, which makes it hard for an eavesdropper to deduce the shared key even if they intercept the exchanged values.

2. **Public Parameters**:

   **Prime Number (p)**: A large prime number is agreed upon by both parties.

   **Base (g)**: A number less than the prime (called the generator), which has special mathematical properties related to modular arithmetic.

These values (p and g) are public and can be known to anyone, including potential attackers.

3. **Private Keys**:

   Each party generates their **own private key** (a secret number) that is never shared with anyone else.

   - Let's say Alice's private key is **a**, and Bob's private key is **b**.

4. **Public Keys**:

   Each party computes their **public key** based on the generator, prime number, and their own private key:

   - Alice computes her public key as **A = g^a mod p**.
   - Bob computes his public key as **B = g^b mod p**.

   These public keys (**A** and **B**) are exchanged between Alice and Bob.

5. **Shared Secret**:

   After exchanging public keys, each party can compute a **shared secret key**:

   - Alice computes **S = B^a mod p**.
   - Bob computes **S = A^b mod p**.

   Both Alice and Bob will end up with the same shared secret **S** because of the properties of modular arithmetic:

   - Bamod p=(gb)amod p=gabmod pB^a \mod p = (g^b)^a \mod p = g^{ab} \mod pBamodp=(gb)amodp=gabmodp
   - Abmod p=(ga)bmod p=gabmod pA^b \mod p = (g^a)^b \mod p = g^{ab} \mod pAbmodp=(ga)bmodp=gabmodp

   This shared secret **S** can now be used as the key for symmetric encryption to secure further communication.

6. **Security**:

   While the public values AAA, BBB, ggg, and ppp are transmitted over the insecure channel, the private keys **a** and **b** are never shared.

   An attacker, knowing only ggg, ppp, AAA, and BBB, would have to solve the discrete logarithm problem to figure out **a** or **b**, which is computationally infeasible for sufficiently large values of ppp.

**Example of Diffie-Hellman Key Exchange:**

1. **Public values**: p=23p = 23p=23, g=5g = 5g=5.
2. **Alice's private key**: a=6a = 6a=6, computes public key A=56mod 23=8A = 5^6 \mod 23 = 8A=56mod23=8.

3. **Bob's private key**: b=15b = 15b=15, computes public key B=515mod 23=19B = 5^{15} \mod 23 = 19B=515mod23=19.
4. **Public keys exchanged**: Alice sends A=8A = 8A=8 to Bob, and Bob sends B=19B = 19B=19 to Alice.
5. **Shared secret**:

   Alice computes S=196mod 23=2S = 19^6 \mod 23 = 2S=196mod23=2.

   Bob computes S=815mod 23=2S = 8^{15} \mod 23 = 2S=815mod23=2.

Now, both Alice and Bob share the secret value S=2S = 2S=2, which can be used as a key for symmetric encryption.

**Advantages:**

- **No need to share a secret key upfront**: Unlike symmetric encryption, parties do not need to pre-share a secret key, making it more scalable and secure.

**Drawbacks:**

- **Vulnerable to man-in-the-middle attacks**: If an attacker intercepts and alters the public keys during exchange, they can trick both parties into establishing shared secrets with them rather than with each other. To prevent this, DHKE is often combined with authentication mechanisms (e.g., digital signatures).

**Modern Uses:**

Diffie-Hellman is commonly used in modern security protocols such as **TLS/SSL** (for secure internet connections), **VPNs**, and secure messaging apps. Variants of the protocol, such as **Elliptic Curve Diffie-Hellman (ECDH)**, use elliptic curves to provide stronger security with smaller key sizes.

**Elliptic Curve Cryptography (ECC)** is a form of public key cryptography based on the mathematics of elliptic curves. ECC provides the same level of security as traditional cryptographic systems like RSA but with much smaller key sizes, making it more efficient in terms of processing power and memory usage.

**Key Concepts of Elliptic Curve Cryptography:**

1. **Elliptic Curve Equation:** An elliptic curve is defined by an equation of the form:

y2=x3+ax+by^2 = x^3 + ax + by2=x3+ax+b

where aaa and bbb are constants, and the equation describes a curve in a two-dimensional plane.

For ECC, this equation is typically used over a finite field, meaning the variables and results are confined to a fixed set of numbers (usually a prime number of points or binary numbers).

2. **Elliptic Curve Group:**

The points on the curve, combined with a special point called the **point at infinity**, form an **abelian group**. In this group, a set of mathematical operations can be performed, like **point addition** and **scalar multiplication**.

The group operation is the addition of two points on the elliptic curve. Scalar multiplication refers to multiplying a point on the curve by a scalar (integer), which is central to ECC operations.

3. **Public and Private Keys:**

   The private key is a random number (integer), while the public key is a point on the elliptic curve derived from multiplying the base point (a predefined point on the curve) by the private key.

   If the private key is kkk and the base point is GGG, the public key PPP is: P=kGP = kGP=kG

   **G** is a point that is publicly known, and the result of multiplying **G** by **k** gives a point on the curve, **P**, which is the public key.

4. **Elliptic Curve Discrete Logarithm Problem (ECDLP):**

   The security of ECC relies on the difficulty of the **Elliptic Curve Discrete Logarithm Problem**. Given a point PPP and a base point GGG, it is computationally infeasible to determine the private key kkk from the public key P=kGP = kGP=kG.

   This problem is harder to solve than the integer factorization problem on which RSA is based, allowing ECC to provide equivalent security with much smaller key sizes.

5. **Smaller Key Sizes with Strong Security:**

   ECC provides the same level of security as RSA but with significantly smaller key sizes, which makes ECC more efficient. For example:

   A 256-bit ECC key is as secure as a 3072-bit RSA key.

   A 384-bit ECC key is as secure as a 7680-bit RSA key

**Advantages of ECC:**

1. **Efficiency:**

   ECC requires much smaller key sizes compared to other public key cryptosystems like RSA, resulting in faster computations and lower resource usage (e.g., CPU, memory, bandwidth).

2. **Stronger Security at Smaller Key Sizes:**

   ECC achieves the same level of security with shorter keys, making it more scalable and practical for devices with limited resources (like smartphones, IoT devices).

3. **Performance:**

Due to its smaller key sizes and faster computations, ECC is well-suited for secure communications in constrained environments such as mobile devices, wireless communications, and embedded systems.

Message authentication and hash functions are crucial components in cryptographic systems for ensuring data integrity and authenticity. Here's a quick overview of each:

**Hash Functions:**

1. **Definition**: A hash function takes an input (or "message") and returns a fixed-size string of bytes. The output is typically a "digest" that uniquely represents the input data.
2. **Properties**:
    - **Deterministic**: The same input always produces the same output.
    - **Fast Computation**: Efficiently computes the hash value.
    - **Pre-image Resistance**: Hard to reverse the hash to retrieve the original input.
    - **Collision Resistance**: Hard to find two different inputs that produce the same hash.
    - **Avalanche Effect**: A small change in input drastically changes the output hash.
3. **Common Algorithms**: MD5, SHA-1, SHA-256, SHA-3.

**Message Authentication:**

1. **Definition**: Ensures that a message comes from a legitimate source and has not been altered during transmission.
2. **Techniques**:

    **Message Authentication Code (MAC)**: A short piece of information used to authenticate a message and confirm its integrity and authenticity. It uses a secret key along with the message.

    **HMAC (Hash-based MAC)**: Combines a cryptographic hash function with a secret key. Commonly used algorithms include HMAC-SHA256.

    **Digital Signatures**: Use asymmetric cryptography to provide authentication. The sender signs the message with their private key, and the receiver can verify it with the sender’s public key. Common algorithms include RSA, DSA, and ECDSA.

3. **Applications**:

    **Secure communications**: Ensuring that messages sent over a network are not tampered with.

    **Data integrity**: Verifying that files or data have not been altered.

**Authentication Requirements**

1. **Confidentiality**:

Ensure that sensitive information (e.g., credentials) is not exposed to unauthorized entities during authentication.

Typically achieved through encryption.

2. **Integrity**:

   The authentication data (such as passwords, tokens, or biometric data) must not be altered during transmission.

   Ensured by using cryptographic methods like message integrity codes or hash functions.

3. **Freshness**:

   Prevents replay attacks where an old authentication message is reused by an attacker. This can be handled using nonces, timestamps, or session tokens.

4. **Mutual Authentication**:

   Both parties (client and server) should verify each other's identity.

   Common in protocols like TLS, where both the server and client may present certificates.

5. **Scalability**:

   The authentication system should be able to handle a large number of users and transactions without performance degradation.

6. **Authorization**:

   After authentication, the system must enforce the appropriate access control policies to ensure that authenticated users can only access what they are permitted to.

7. **Non-repudiation**:

   Ensures that a user cannot deny actions that they have performed. This is often achieved through digital signatures.

**Authentication Functions**

1. **Password-based Authentication**:

   **Description**: The most common form, where users present a password that is checked against stored credentials.

   **Strength**: Depends on the complexity of the password and the protection of the stored hash (e.g., through salt and hashing).

   **Weakness**: Susceptible to brute-force attacks, phishing, or poor password hygiene.

2. **Token-based Authentication**:

**Description**: Users authenticate using tokens (e.g., session tokens, JWTs, one-time passwords).

**Types**:

**Session Tokens**: After login, the server provides a token that the client sends with each request.

**One-time Passwords (OTP)**: Temporary, single-use passwords for additional security (often used in two-factor authentication).

- **Strength**: Tokens are short-lived and more secure than passwords alone.

3. **Biometric Authentication**:

   **Description**: Uses unique biological traits like fingerprints, facial recognition, or retinal scans to authenticate.

   **Strength**: Difficult to forge.

   **Weakness**: Privacy concerns, potential for spoofing with advanced methods.

4. **Multi-factor Authentication (MFA)**:

   **Description**: Combines two or more authentication factors:

   **Something you know** (password).

   **Something you have** (a token, smartphone).

   **Something you are** (biometric).

   - **Strength**: Reduces the risk of a single point of failure.

5. **Certificate-based Authentication**:

   **Description**: Involves digital certificates (e.g., X.509 certificates) for authentication. Used in SSL/TLS for securing web traffic.

   **Strength**: High security, used in mutual authentication for systems like HTTPS.

   **Weakness**: Complexity in management and distribution of certificates.

6. **Cryptographic Authentication**:

   **Description**: Involves using cryptographic keys for authentication, such as:

   **Public Key Infrastructure (PKI)**: Uses public/private key pairs for authentication, such as in digital signatures.

   **Message Authentication Codes (MAC)**: Ensure data authenticity by using a shared secret key between the communicating parties.

   **Strength**: Provides a strong level of security.

7. **Challenge-Response Authentication**:

**Description**: A server sends a challenge (e.g., random number) to the user, who must return a valid response (often involving some cryptographic calculation).

**Strength**: Prevents replay attacks and ensures freshness.

**Weakness**: Requires secure storage and management of cryptographic keys.

**Hash Functions**

1. **Definition**: A **hash function** is a mathematical algorithm that transforms an arbitrary-length input (message) into a fixed-size output, known as a hash value or digest.
2. **Properties**:
    - **Deterministic**: The same input always produces the same output.
    - **Fixed Output Length**: Regardless of the input size, the output (hash) is always of a fixed length (e.g., 256 bits for SHA-256).
    - **Pre-image Resistance**: Given a hash, it should be computationally infeasible to find the original input (message).
    - **Collision Resistance**: It should be infeasible to find two different inputs that produce the same hash.
    - **Second Pre-image Resistance**: Given a specific input and its hash, it should be difficult to find a different input that produces the same hash.
    - **Avalanche Effect**: A small change in the input drastically changes the output hash.
3. **Common Hash Functions**:
    - **MD5**: Produces a 128-bit hash value but is considered broken due to vulnerabilities in collision resistance.
    - **SHA-1**: Produces a 160-bit hash but has also been deprecated due to collision vulnerabilities.
    - **SHA-2**: A family of hash functions (SHA-224, SHA-256, SHA-384, and SHA-512) widely used in modern cryptography.
    - **SHA-3**: A newer hash function standardized in 2015 as an alternative to SHA-2, using the Keccak algorithm.
4. **Applications**:
    - **Data Integrity**: Verifying that data has not been altered (e.g., file checksums).
    - **Digital Signatures**: Hashing the data before signing it with a private key.
    - **Password Hashing**: Storing hashed passwords in databases for secure authentication.

**Message Authentication Codes (MACs)**

1. **Definition**: A **Message Authentication Code (MAC)** is a short piece of information derived from a message and a secret key, designed to authenticate the integrity and authenticity of the message.
2. **Purpose**:
    - **Integrity**: Ensures that the message has not been tampered with.
    - **Authentication**: Ensures that the message originates from the intended sender (who possesses the secret key).
3. **How MACs Work**:
    - The sender and receiver share a secret key.
    - The sender computes the MAC by applying a cryptographic function to the message and the secret key.
    - The MAC is sent along with the message.
    - The receiver, who also possesses the secret key, computes the MAC independently on the received message and compares it to the received MAC.
    - If the two MACs match, the message is authenticated and verified.

**Differences Between Hash Functions and MACs:**

| ASPECT | Hash Function Message | Authentication Code (MAC) |
|---|---|---|
| Key Usage | No secret key required | Requires a secret key shared between parties |
| Purpose | Ensures data integrity (collision resistance) | Ensures both data integrity and authenticity |
| Attack Resistance | Vulnerable to attacks like collision attacks | Resistant to forgery without knowing the secret key |
| Common Use Cases | Data integrity checks, password hashing | Message authentication in network protocols (TLS) |

**MAC Algorithms SHA-512:**

The SHA-512 algorithm is part of the SHA-2 family of cryptographic hash functions. It's designed to produce a fixed-size 512-bit hash value from an arbitrary input. When used in the construction of Message Authentication Codes (MACs), it combines the security of a hash function with a secret key to provide both message integrity and authenticity.

**How SHA-512 Works**

1. **Message Preprocessing**:
    - The message is divided into 1024-bit blocks, padded with extra bits so that the total message length is a multiple of 1024.

- The padding includes a 1 bit followed by enough 0 bits and ends with a 128-bit representation of the message length (since SHA-512 processes 1024-bit blocks).

2. **Message Digest Computation**:
    - The algorithm initializes eight 64-bit words, known as the initial hash value.
    - For each 1024-bit block, a series of operations (including bitwise shifts, rotations, and modular additions) are applied to the message and hash values.
    - These operations mix the bits of the message block with the current hash value, updating it iteratively.
3. **Final Hash**:
    - After processing all message blocks, the final hash value is obtained by concatenating the updated values of the eight 64-bit words. This final value is the 512-bit hash output.

**SHA-512 in HMAC (HMAC-SHA-512)**

When **SHA-512** is used to create an HMAC (Hash-based Message Authentication Code), it follows this process:

1. **Secret Key**: A secret key K shared between the sender and receiver is required. If the key is longer than 1024 bits (SHA-512's block size), it is hashed using SHA-512 to reduce its length. If the key is shorter than 1024 bits, it is padded with zeros to the block size.
2. **HMAC-SHA-512 Steps**:
    - The key is XORed with two fixed constants: an **inner padding** (ipad) and an **outer padding** (opad).
    - First, the inner hash is calculated: H(K_ipad || message), where K_ipad is the key XORed with the ipad and message is the original message.
    - Then, the outer hash is computed: H(K_opad || inner_hash), where K_opad is the key XORed with the opad and inner_hash is the result from the first hash.
    - The final HMAC-SHA-512 result is this outer hash.

**Security of HMAC-SHA-512**

- **Resistance to Length Extension Attacks**: HMAC-SHA-512 prevents length-extension attacks, which are possible with plain hash functions like SHA-512. This is because the key and message are processed in a way that makes it impossible for attackers to modify or append to the message without knowing the secret key.
- **Collision Resistance**: SHA-512 has a large output size (512 bits), making it highly resistant to collision attacks. Collisions, where two different inputs produce the same hash output, are computationally infeasible due to the large hash size.

**Advantages of Using SHA-512 for HMAC:**

1. **High Security**: SHA-512 provides a higher level of security compared to shorter hash functions like SHA-256 due to its longer hash output (512 bits).
2. **Efficiency on 64-bit Systems**: SHA-512 is optimized for 64-bit processors, making it relatively fast and efficient on modern hardware.
3. **Longer Key Size**: It allows for larger key sizes, making brute-force attacks more difficult.
4. **Robustness**: It's highly resistant to most cryptographic attacks, making it suitable for environments that require high security, like financial transactions, SSL/TLS encryption, and more.

**HMAC-SHA-512 Applications**

- **TLS/SSL**: Ensures the integrity and authenticity of messages during secure communications.
- **IPsec**: Used in VPNs and other network security protocols to protect the authenticity of transmitted data.
- **Digital Signatures**: In combination with private keys, HMAC-SHA-512 can secure digital signatures.
- **Password Storage**: Used for securely storing and verifying passwords in authentication systems.

**Example of HMAC-SHA-512 Calculation**

Given a message "Hello, World!" and a key, the HMAC-SHA-512 calculation follows these steps:

1. **Key Preparation**:
    - If the key is shorter than 1024 bits, it is padded. If it's longer, it's hashed to fit the block size.
2. **Inner and Outer Padding**:
    - Apply the inner padding (XOR with the ipad), then hash the result along with the message.
    - Apply the outer padding (XOR with the opad), then hash the result with the inner hash.
3. **Final Output**:
    - The final HMAC value is a 512-bit digest, providing both integrity and authenticity of the message.