Unit II

Fundamentals of Algorithms in Cybersecurity

Q1. Explain Data Encryption Standard (DES) and Rivest-Shamir-Adleman (RSA) Algorithms.

Ans: Details of the **Data Encryption Standard (DES)** and the **Rivest-Shamir-Adleman (RSA)** algorithms:

1. **Data Encryption Standard (DES)**:
   - **Overview**:
     - DES is a symmetric-key block cipher used for secure data transmission.
     - Developed by IBM in the 1970s, it was widely adopted as a standard encryption algorithm.
     - DES operates on 64-bit blocks of data and uses a 56-bit key.
   - **Key Features**:
     - **Symmetric**: Same key is used for both encryption and decryption.
     - **Block Cipher**: Processes fixed-size blocks of data.
     - **Substitution-Permutation Network (SPN)**: Utilizes substitution (S-boxes) and permutation (P-boxes) operations.
     - **Feistel Structure**: Divides the block into two halves and applies multiple rounds of transformation.
   - **Limitations**:
     - **Key Length**: 56-bit key length is considered insufficient for modern security.
     - **Vulnerable to Brute Force Attacks**: Due to the limited key space.
   - **Status Today**:
     - DES is largely obsolete due to its vulnerabilities.
     - Triple DES (3DES) is a variant that applies DES three times with different keys for improved security.
2. **Rivest-Shamir-Adleman (RSA) Algorithm**:
   - **Overview**:
     - RSA is a widely used asymmetric (public-key) cryptosystem.
     - Developed by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977.
     - Based on the mathematical difficulty of factoring large semiprime numbers.
   - **Key Features**:
     - **Public/Private Key Pair**: Encryption key is public, while decryption key is private.

        - **Modular Exponentiation**: Core operation for encryption and decryption.
        - **Digital Signatures**: RSA is used for signing messages.
    - **Usage**:
        - Secure data transmission, digital signatures, and key exchange.
    - **Example (Python)**:
    - # Generate RSA keys
    - public_key, private_key = generate_rsa_keys(512)
    - message = "HELLO"
    - encrypted_message = encrypt(message, public_key)
    - decrypted_message = decrypt(encrypted_message, private_key)
    - print("Original Message:", message)
    - print("Encrypted Message:", encrypted_message)
    - print("Decrypted Message:", decrypted_message)

      Output:

      Original Message: HELLO
      Encrypted Message: [343, 466, 125, 125, 141]
      Decrypted Message: HELLO

    - **Security**:
        - RSA's security relies on the difficulty of factoring large semiprime numbers.
        - Key length matters: longer keys enhance security.

In summary, DES is a symmetric-key cipher with limitations, while RSA is an asymmetric algorithm used for secure communication and digital signatures.

Q2. Explain Diffie-Hellman Key Exchange Algorithm with an Example

Ans: Certainly! Let's dive into the **Diffie-Hellman key exchange algorithm** and illustrate it with an example:

1. **Overview**:
    - Diffie-Hellman (DH) allows two parties to securely establish a shared secret over an insecure channel.
    - It's a fundamental building block for secure communication and key exchange.
    - DH is based on modular exponentiation and relies on the difficulty of the discrete logarithm problem.
2. **Steps**:
    - Alice and Bob agree on two large prime numbers, p and g.
    - They also choose a public key exchange algorithm.
    - Individual steps:
        - Alice chooses a secret integer, a, and computes (A = g^a \mod p). She sends A to Bob.

        - Bob chooses a secret integer, b, and computes (B = g^b \mod p). He sends B to Alice.
        - Both Alice and Bob now have (A) and (B).
        - The shared secret key is computed as:
            - Alice: (s = B^a \mod p)
            - Bob: (s = A^b \mod p)
3. **Example**:
    - Let's say Alice and Bob want to establish a shared secret.
    - They agree on p = 23 and g = 5.
    - Alice chooses a = 6:
        - (A = 5^6 \mod 23 = 8)
    - Bob chooses b = 15:
        - (B = 5^{15} \mod 23 = 19)
    - Alice computes:
        - (s = 19^6 \mod 23 = 2)
    - Bob computes:
        - (s = 8^{15} \mod 23 = 2)
    - Now both Alice and Bob share the secret key 2.
4. **Security**:
    - DH is secure because calculating the discrete logarithm (finding a or b) is computationally hard.
    - It ensures confidentiality during key exchange.

In summary, Diffie-Hellman enables secure key exchange even when parties haven't met beforehand. It's a crucial component of modern cryptography.

Q3: Explain Digital Signature Algorithm (DSA) With an Example.

Ans: Certainly! Let's explore the **Digital Signature Algorithm (DSA)** and illustrate it with an example:

1. **Overview**:
    - DSA is a public-key cryptographic technique used for creating and verifying digital signatures.
    - It ensures the authenticity and integrity of messages.
    - Unlike encryption algorithms, DSA is specifically designed for digital signatures.
2. **How DSA Works**:
    - **Key Generation**:
        - Alice generates a pair of keys:
            - **Private Key (PRa)**: Kept secret.
            - **Public Key (PUa)**: Shared with others.
    - **Signing Process**:
        - When Alice wants to sign a message (e.g., a document), she follows these steps:

        1. Computes a hash of the message (using a hash function).
        2. Encrypts the hash using her private key (PRa) to create the signature.
        3. Attaches the signature to the message.
    - **Verification Process** (by Bob, the recipient):
        1. Bob receives the message and the attached signature.
        2. Computes the hash of the received message.
        3. Decrypts the signature using Alice's public key (PUa).
        4. If the decrypted signature matches the computed hash, the message is authentic.
3. **Example**:
    - Suppose Alice wants to sign the message "CONFIDENTIAL."
    - She computes the hash (e.g., SHA-256) of the message: H(CONFIDENTIAL).
    - Encrypts the hash using her private key: Signature = Encrypt(PRa, H(CONFIDENTIAL)).
    - Bob receives the message and signature.
    - Bob computes H(CONFIDENTIAL) and decrypts the signature using Alice's public key.
    - If the decrypted signature matches the computed hash, Bob knows the message is authentic.
4. **Security**:
    - DSA relies on the difficulty of solving discrete logarithm problems.
    - It ensures non-repudiation (Alice cannot deny signing the message).

In summary, DSA provides a secure way to verify the authenticity of messages using digital signatures.

Q4. Explain the Following Types of One-time Password (OTP) Algorithms with Examples:

a. Time-based OTP (TOTP)

b. HMAC-based OTP (HOTP)

Ans: Certainly! Let's explore the two types of One-Time Password (OTP) algorithms: Time-based OTP (TOTP) and HMAC-based OTP (HOTP), along with examples:

1. HMAC-based OTP (HOTP):
    - Overview:
        - HOTP uses a counter-based moving factor.
        - The seed (secret key) remains static, but each time the HOTP is requested, the moving factor increments based on a counter.
        - It relies on the Hash-based Message Authentication Code (HMAC) using the SHA-1 hash function.
    - Example:
        - Alice and Bob share a secret key (seed).
        - When Alice logs in, the server calculates the HOTP using the counter value.

        - If the calculated HOTP matches the one Alice provides, she gains access.
        - Each validated HOTP increments the counter for the next login attempt.
2. Time-based OTP (TOTP):
    - Overview:
        - TOTP uses a time-based moving factor.
        - The seed is static (like HOTP), but the moving factor changes based on time intervals (typically 30 or 60 seconds).
        - It's commonly used for two-factor authentication (2FA) via mobile apps.
    - Example:
        - Alice's phone generates a TOTP based on the current time.
        - She enters the TOTP along with her regular password during login.
        - The server validates the TOTP against the expected value.
        - If they match, Alice gains access.

In summary, HOTP relies on a counter, while TOTP uses time intervals for generating one-time passwords. Both enhance security by requiring additional authentication beyond regular passwords.