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

A. **Data Encryption Standard (DES):** It is a symmetric-key encryption technique used for the encryption of electronic data.

Key Features:

- Symmetric Key Algorithm: DES uses the same key for both encryption and decryption.
- Block Cipher: It operates on fixed-size blocks of data, specifically 64 bits.
- Key Size: DES uses a 56-bit key for encryption. The actual key is 64 bits long, but 8 bits are used for parity checking.
- Feistel Structure: DES employs a Feistel network, which divides the block of plaintext into two halves and processes them through multiple rounds of permutation and substitution.
- Rounds: DES consists of 16 rounds of processing. Each round involves expansion, permutation, substitution (using S-boxes), and XOR with a subkey derived from the main key.

Encryption Process:

- Initial Permutation (IP): The 64-bit plaintext block is permuted according to a fixed table.
- Round Function (F-function): For each of the 16 rounds, the right half of the block is expanded, mixed with a round key, passed through S-boxes, and permuted. The output is XORed with the left half.
- Swap: The halves are swapped (except in the last round).
- Final Permutation (FP): The inverse of the initial permutation is applied to the combined halves to produce the ciphertext.

Decryption Process:

Decryption follows the same steps as encryption but in reverse order, using the same key.

Security Concerns:

- Key Size: The 56-bit key is considered weak by modern standards, making DES vulnerable to brute-force attacks.
- Alternative Algorithms: DES has been largely replaced by the Advanced Encryption Standard (AES), which provides better security and efficiency.

Rivest-Shamir-Adleman (RSA) Algorithm: It is a public-key encryption technique that is widely used for secure data transmission.

Key Features:

- Asymmetric Key Algorithm: RSA uses a pair of keys, a public key for encryption, and a private key for decryption.
- Based on Number Theory: The security of RSA relies on the difficulty of factoring large composite numbers, which is a well-known problem in number theory.
- Key Size: RSA keys are typically 2048 bits or longer to ensure security.

Key Generation:

1. Generate Two Large Primes: Select two distinct large prime numbers, p and q.

2. Compute Modulus n: $n=p \times q$.

3. Compute Euler's Totient $\phi(n)$: $\phi(n)=(p-1) \times (q-1)$.

4. Select Public Exponent e: Choose an integer e such that $1<e<\phi(n)$ and gcd $(e, \phi(n)) = 1$.

5. Compute Private Exponent d: Determine d as the modular multiplicative inverse of e modulo $\phi(n)$, i.e., $d \times e \equiv 1 (\text{mod } \phi(n))$.

Encryption Process:

1. Public Key: Consists of (e, n).

2. Encryption: Convert the plaintext message M into an integer m such that $0 \leq m < n$. The ciphertext c is computed as: $c = m \text{ ^ } e \pmod n$

Decryption Process:

1. Private Key: Consists of (d, n).

2. Decryption: Convert the ciphertext c back into plaintext by computing: $m = c \text{ ^ } d \pmod n$

Security:

- The security of RSA is based on the computational difficulty of factoring the large integer n into its prime components p and q.
- RSA is secure as long as the key sizes are sufficiently large (typically 2048 bits or more).

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

A. **Diffie-Hellman Key Exchange Algorithm:** It is a method for securely exchanging cryptographic keys over a public channel. It allows two parties to establish a shared secret key, which can be used for subsequent encrypted communication, even if they have never met or exchanged keys beforehand. The security of the algorithm relies on the difficulty of the discrete logarithm problem.

Key Concepts:

1. Public Parameters:

- A large prime number p.
- A primitive root modulo p, often denoted as g.

2. Private Keys:

- Each party chooses a private key, which is a secret integer.
- Let's denote the private keys of the two parties as a and b.

3. Public Keys:

- Each party computes their public key by raising the primitive root g to the power of their private key modulo p.
- The public keys are exchanged over the public channel.

4. Shared Secret:

- Each party computes the shared secret by raising the received public key to the power of their private key modulo p.
- Both parties arrive at the same shared secret, which can be used as a symmetric key for encryption.

Example

Let's walk through an example with simple numbers to illustrate the Diffie-Hellman key exchange.

1. Public Parameters:

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

2. Private Keys:

- Alice chooses her private key a=6.
- Bob chooses his private key b=15.

3. Public Keys:

- Alice computes her public key: A=g ^ a mod p=5^6 mod23=15625mod23=8.
- Bob computes his public key: B=g ^ b mod p=5^15 mod23=30517578125mod23=19.

4. Exchange Public Keys:

- Alice sends A=8 to Bob.
- Bob sends B=19 to Alice.

5. Compute Shared Secret:

- Alice computes the shared secret: s=B ^ a mod p=19^6 mod23=47045881mod23=2.
- Bob computes the shared secret: s=A ^ b mod p=8^15 mod23=35184372088832mod23=2.

Both Alice and Bob now have the shared secret s=2, which can be used as a key for symmetric encryption.

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

A. **Digital Signature Algorithm (DSA):** It is a Federal Information Processing Standard for digital signatures. It is used to generate a digital signature that can be used to verify the authenticity and integrity of a message. DSA is based on the mathematical principles of modular exponentiation and discrete logarithms, similar to the Diffie-Hellman key exchange.

Key Concepts

1. Key Generation:

- Prime Number (p): A large prime number.
- Subprime Number (q): A 160-bit prime factor of p−1.
- Base (g): An element of high order modulo p, typically g=h^(p−1)/q mod p, where h is any integer such that 1<h<p−1 and h^(p−1)/q nottripleequalto1modp.
- Private Key (x): A randomly chosen integer x such that 0<x<q.
- Public Key (y): Computed as y=g ^ x mod p.

2. Signature Generation:

- Message (m): The message to be signed.
- Hash Function: A hash function (e.g., SHA-1) to generate a hash value of the message.
- Random Number (k): A randomly chosen integer k such that 0<k<q.
- Signature Components (r, s):
  - r = (g ^ k mod p) mod q.
  - s = (k^−1 ·(H(m)+x ·r)) mod q.

3. Signature Verification:

- Verification Components:
  - Compute w=s^−1 mod q.
  - Compute u1=(H(m)·w) mod q.
  - Compute u2= (r · w) mod q.
  - Compute v= ((g^u1 ·y^u2) mod p) mod q.
- The signature is valid if and only if v=r.

Example:

1. Key Generation:

- Prime number p=23.
- Subprime number q=11 (factor of p−1).
- Base g=2.
- Private key x=3.
- Public key y=g ^ x mod p=2^3 mod23=8.

2. Signature Generation:

- Message m="Hello".
- Hash of the message H(m)=5 (simplified for this example).
- Random number k=6.
- Compute r = (g ^ k mod p) mod q= (2^6 mod 23) mod11=64mod23mod11=18mod11=7.
- Compute s= (k^−1 ·(H(m)+x · r)) mod q= (6^−1 · (5+3·7)) mod11.
  - k^−1 mod q=2 (since 6·2mod11=1).
  - s= (2· (5+21)) mod11= (2·26) mod11=52mod11=8.
- Signature: (r, s) = (7,8).

Signature Verification:

- Compute w=s^−1 mod q=8^−1 mod11=7 (since 8·7mod11=1).
- Compute u1=(H(m)·w) mod q= (5·7) mod11=35mod11=2.
- Compute u2= (r ·w) mod q= (7·7) mod11=49mod11=5.
- Compute v= ((g^u1 ·y^u2) mod p) mod q= ((2^2 ·8^5) mod23) mod11.
  - 2^2 mod23=4.
  - 8^5 mod23=32768mod23=16.
  - v=(4·16)mod23=64mod23=18mod11=7.
- Verify v=r. Since v=7 and r=7, the signature is valid.

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

a. Time-based OTP (TOTP)

b. HMAC-based OTP (HOTP)

A. **One-Time Password (OTP) Algorithms:** OTPs are temporary, single-use passwords that help enhance security by reducing the risk of replay attacks. Two widely used OTP algorithms are Time-based OTP (TOTP) and HMAC-based OTP (HOTP).

a. Time-based OTP (TOTP): It is an extension of the HOTP algorithm that introduces the concept of time-based expiration. It generates a one-time password that is valid only for a short time window,

typically 30 seconds. This ensures that the OTP is only valid for a limited period, reducing the risk of it being intercepted and used maliciously.

Working:

1. Shared Secret: A secret key is shared between the server and the client.
2. Current Time: The current time is used as a moving factor.
3. Time Step: A predefined time step (e.g., 30 seconds) defines the time window for which the OTP is valid.
4. HMAC Calculation: The current time step is combined with the shared secret and processed through an HMAC-SHA1 algorithm.
5. Truncate and Modulo: The result is truncated and a modulo operation is applied to get the OTP of the desired length.

Example:

Assuming a shared secret 'SECRET' and a time step of 30 seconds:

1. Shared Secret: 'SECRET'
2. Current Time: Let's say the current Unix time is 1618329600 (April 13, 2021, 00:00:00 UTC).
3. Time Step: 30 seconds.
4. Time Counter: Counter=floor (1618329600/30) = 53944320
5. HMAC Calculation: Calculate'HMAC-SHA1(SECRET, 53944320)'.
6. Truncate and Modulo: Truncate the HMAC result and apply modulo operation to get a 6-digit OTP, for example, 123456.

b. HMAC-based OTP (HOTP): It is an OTP algorithm based on HMAC (Hash-based Message Authentication Code). It generates OTPs based on a counter value, which increments with each new OTP. Unlike TOTP, HOTP does not rely on time but on an event counter, making it suitable for scenarios where synchronization can be ensured between the server and the client.

Working:

1. Shared Secret: A secret key is shared between the server and the client.
2. Counter: A counter value that increments with each OTP generation.
3. HMAC Calculation: The counter and the shared secret are combined and processed through an HMAC-SHA1 algorithm.
4. Truncate and Modulo: The result is truncated and a modulo operation is applied to get the OTP of the desired length.

Example:

Assuming a shared secret 'SECRET' and a counter value of 1:

1. Shared Secret: 'SECRET'
2. Counter: 1
3. HMAC Calculation: Calculate HMAC-SHA1(SECRET, 1).
4. Truncate and Modulo: Truncate the HMAC result and apply modulo operation to get a 6-digit OTP, for example, 654321.