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

Data Encryption Standard (DES):

1. DES is a symmetric-key algorithm, meaning the same key is used for both encryption and decryption.

2. It was developed by IBM in the 1970s and adopted as a federal standard in the United States in 1977.

3. DES operates on 64-bit blocks of data. It takes a 64-bit plaintext and transforms it into a 64-bit ciphertext.

4. The key used in DES is 56 bits long. Although the key is technically 64 bits, 8 bits are used for parity checks, leaving an effective key length of 56 bits.

5. DES uses a series of complex permutations and substitutions based on the key. The algorithm consists of 16 rounds of processing, each round using a different subkey derived from the main key.

6. Despite its initial security, DES has become vulnerable to brute-force attacks due to advances in computational power, leading to its gradual replacement by more secure algorithms like AES (Advanced Encryption Standard).

Rivest-Shamir-Adleman (RSA):

1. RSA is an asymmetric-key algorithm, meaning it uses a pair of keys: a public key for encryption and a private key for decryption.

2. It was invented in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman at the Massachusetts Institute of Technology.

3. RSA is based on the mathematical difficulty of factoring large composite numbers. The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers.

4. The key generation process involves selecting two large prime numbers and multiplying them to produce a modulus. The public key is derived from the modulus and a chosen public exponent, while the private key is derived from the modulus and a private exponent.

5. RSA can encrypt a block of data by raising it to the power of the public exponent modulo the product of the primes. Decryption involves raising the ciphertext to the power of the private exponent modulo the product of the primes.

6. RSA is widely used for secure data transmission, digital signatures, and key exchange. However, it is computationally intensive compared to symmetric-key algorithms, making it slower for encrypting large amounts of data. It is often used in combination with symmetric-key algorithms for efficiency.

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

The Diffie-Hellman Key Exchange is a method used to securely exchange cryptographic keys over a public channel. This algorithm allows two parties to generate a shared secret key, which can then be used for secure communication.

1. Public Parameters: Two large numbers are publicly agreed upon:

   - A prime number p
   - A base (also called a generator) g, where $1 < g < p$

2. Private Keys: Each party generates their own private key:

   - Party A chooses a private key a (a random number kept secret)
   - Party B chooses a private key b (another random number kept secret)

3. Public Keys: Each party computes their public key using the public parameters and their private key:

   - Party A computes $A = g^a \mod p$
   - Party B computes $B = g^b \mod p$

4. Exchange Public Keys: The public keys A and B are exchanged between the two parties over the public channel.

5. Compute Shared Secret: Each party computes the shared secret using their private key and the other party's public key:

- Party A computes the shared secret $S_A = B^a \mod p$
- Party B computes the shared secret $S_B = A^b \mod p$

Since $B^a \mod p$ is equal to $A^b \mod p$, both parties arrive at the same shared secret S.

1. Public Parameters:
   - Prime number $p = 23$
   - Base $g = 5$

2. Private Keys:
   - Party A's private key $a = 6$
   - Party B's private key $b = 15$

3. Public Keys:
   - Party A computes $A = 5^6 \mod 23$
     - $5^6 = 15625$
     - $15625 \mod 23 = 8$
     - So, $A = 8$
   - Party B computes $B = 5^{15} \mod 23$
     - $5^{15} = 30517578125$
     - $30517578125 \mod 23 = 19$
     - So, $B = 19$

4. Exchange Public Keys:
   - Party A sends $A = 8$ to Party B

- Party B sends $B = 19$ to Party A

5. Compute Shared Secret:

- Party A computes $S_A = 19^6 \mod 23$
  - $19^6 = 47045881$
  - $47045881 \mod 23 = 2$
  - So, $S_A = 2$
- Party B computes $S_B = 8^{15} \mod 23$
  - $8^{15} = 35184372088832$
  - $35184372088832 \mod 23 = 2$
  - So, $S_B = 2$

Both parties have arrived at the same shared secret $S = 2$.

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

1. Key Generation:

- Public Parameters:
  - Choose a prime number p, typically between 1024 and 3072 bits.
  - Choose another prime number q, which is a 160-bit to 256-bit prime divisor of p-1.
  - Choose a number g such that g = h^((p-1)/q) mod p, where h is any integer 1 < h < p-1 and h is chosen such that g > 1.
- Private Key:
  - Choose a private key x such that x is a random integer 0 < x < q.
- Public Key:
  - Compute the public key y such that y = g^x mod p.

2. Signature Generation:

- Given a message m:

- Choose a random integer k such that 0 < k < q (Note: k must be different for each signature).

- Compute r = (g^k mod p) mod q.

- Compute s = (k^(-1) (H(m) + x * r)) mod q, where H(m) is the hash of the message.

- The signature of the message is the pair (r, s).

3. Signature Verification:

- To verify the signature (r, s) of a message m:

- Check that 0 < r < q and 0 < s < q; if not, the signature is invalid.

- 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.

- If v = r, the signature is valid. Otherwise, it is invalid.

Example:

1. Key Generation:

- Choose p = 23 (a small prime for simplicity).

- Choose q = 11 (a prime divisor of p-1).

- Choose g = 4 (such that g = h^((p-1)/q) mod p).

- Choose a private key x = 6.

- Compute the public key y = 4^6 mod 23 = 9.

2. Signature Generation:

- Given a message m (e.g., "Hello"), compute H(m) (let's say H(m) = 5).

- Choose a random k = 3.

- Compute r = (4^3 mod 23) mod 11 = 6.

- Compute s = (3^(-1) (5 + 6 * 6)) mod 11 = (3^(-1) (5 + 36)) mod 11 = (3^(-1) * 41) mod 11. Since 3^(-1) mod 11 = 4, s = (4 * 41) mod 11 = 7.

- The signature is (r, s) = (6, 7).

3. Signature Verification:

- Verify 0 < r < q and 0 < s < q.

- Compute w = 7^(-1) mod 11 = 8.

- Compute u1 = (5 * 8) mod 11 = 40 mod 11 = 7.

- Compute u2 = (6 * 8) mod 11 = 48 mod 11 = 4.

- Compute v = ((4^7 * 9^4) mod 23) mod 11 = (16 * 4) mod 23 mod 11 = 64 mod 23 mod 11 = 18 mod 11 = 7.

- Since v = r, 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).

One-time passwords (OTPs) are passwords that are valid for only one login session or transaction. They are commonly used to enhance security. There are two primary types of OTP algorithms: Time-based OTP (TOTP) and HMAC-based OTP (HOTP).

a. Time-based OTP (TOTP)

Time-based OTP (TOTP) generates a one-time password that changes after a fixed period, typically every 30 seconds.

How TOTP Works:

1. Shared Secret: A shared secret key is established between the client and the server.

2. Current Time: The current time is used as an input.

3. Algorithm: TOTP uses the HMAC-SHA1 algorithm, which takes the secret key and the current time as inputs.

4. Time Step: The current Unix time is divided by a fixed time step (e.g., 30 seconds) to generate a moving factor.

5. Password Generation: The moving factor and the secret key are used to generate a hash. A portion of this hash is then converted into a numeric OTP.

Example:

1. Shared Secret: The shared secret key is "JBSWY3DPEHPK3PXP".

2. Current Time: Suppose the current Unix time is 1625078400 seconds since Epoch.

3. Time Step: Using a 30-second time step, the moving factor is calculated as 1625078400 divided by 30 which equals 54169280.

4. HMAC Calculation: The HMAC-SHA1 hash is generated using the shared secret and the moving factor.

5. Truncate and Convert: The hash is truncated and converted into a 6-digit OTP, for example, 432198.

b. HMAC-based OTP (HOTP)

HMAC-based OTP (HOTP) generates a one-time password based on a counter that is incremented with each new OTP generation.

How HOTP Works:

1. Shared Secret: A shared secret key is established between the client and the server.

2. Counter: A counter value is used as an input. This counter is incremented with each new OTP generation.

3. Algorithm: HOTP uses the HMAC-SHA1 algorithm, which takes the secret key and the counter value as inputs.

4. Password Generation: The counter value and the secret key are used to generate a hash. A portion of this hash is then converted into a numeric OTP.