17 Public-key cryptography

17.1 Introduction

Public-key cryptography, also known as **asymmetric cryptography**, is a cryptographic system that uses a pair of keys:

- A **public key**, which is shared openly.
- A **private key**, which is kept secret by the owner.

Unlike symmetric cryptography, where both the sender and receiver use the same key, public-key cryptography allows secure communication without prior key exchange.

17.2 Basic Concept

In a public-key cryptosystem, encryption and decryption are performed using different keys:

- The sender encrypts a message using the recipient's **public key**.
- The recipient decrypts the message using their **private key**.

The encryption function E and decryption function D satisfy:

$$D(K_{priv}, E(K_{pub}, M)) = M$$

where:

- M is the plaintext message.
- K_{pub} is the public key.
- K_{priv} is the private key.
- $E(K_{pub}, M)$ is the encrypted message (ciphertext).

17.3 Mathematical Foundation

Most public-key cryptosystems rely on mathematical problems that are computationally hard to solve. Commonly used problems include:

- **Integer Factorization Problem**: Used in RSA cryptosystem.
- **Discrete Logarithm Problem**: Used in Diffie-Hellman key exchange and ElGamal cryptosystem.
- **Elliptic Curve Discrete Logarithm Problem**: Used in elliptic curve cryptography (ECC).

17.4 Diffie-Hellman Key Exchange (DHKE)

Diffie-Hellman Key Exchange is an asymmetric cryptographic protocol for key exchange, and its security is based on the computational hardness of solving a discrete logarithm problem.

The Diffie–Hellman key exchange algorithm proceeds as follows:

- **Public Parameter Creation:** A trusted party chooses and publishes a large prime p and an integer g, where g has a large prime order in $\mathbb{F}_p$.
- **Private Computations:**
  - Alice chooses a secret integer a and computes $A \equiv g^a \pmod{p}$.
  - Bob chooses a secret integer b and computes $B \equiv g^b \pmod{p}$.
- **Public Exchange of Values:** Alice sends A to Bob, and Bob sends B to Alice.
- **Further Private Computations:**
  - Alice computes $S_A = B^a = (g^b)^a \pmod{p}$.
  - Bob computes $S_B = A^b = (g^a)^b \pmod{p}$.

  Both Alice and Bob will now have the same shared secret $S = S_A = S_B$, which can be used for further cryptographic operations.

17.4.1 Example of Diffie-Hellman Key Exchange

Let us consider an example where Alice and Bob agree on:

- A prime $p = 23$,
- A generator $g = 5$.

Step 1: Private Key Selection

- Alice chooses a secret integer $a = 6$.
- Bob chooses a secret integer $b = 15$.

Step 2: Compute Public Keys

$$A = g^a \mod p = 5^6 \mod 23 = 15625 \mod 23 = 8.$$

$$B = g^b \mod p = 5^{15} \mod 23 = 30517578125 \mod 23 = 19.$$

Step 3: Exchange Public Keys Alice sends $A = 8$ to Bob, and Bob sends $B = 19$ to Alice.

Step 4: Compute Shared Secret

$$S_A = B^a \mod p = 19^6 \mod 23 = 47045881 \mod 23 = 2.$$

$$S_B = A^b \mod p = 8^{15} \mod 23 = 35184372088832 \mod 23 = 2.$$

Since both Alice and Bob compute the same shared secret $S = 2$, they can now use it for encryption.

17.4.2 Diffie-Hellman Key Exchange Analysis

The security of the Diffie-Hellman key exchange is based on the fact that it is difficult to calculate discrete logarithms in a finite field. An attacker who intercepts the public values A and B would need to solve the discrete logarithm problem to calculate the shared secret key, which is computationally infeasible for large prime numbers.

17.5 ElGamal Cryptosystem

The ElGamal encryption algorithm is a public key cryptography algorithm that uses a key pair consisting of a private key and a public key. The algorithm involves three steps: key generation, encryption, and decryption.

The following are the steps involved in the ElGamal encryption algorithm:

1. **Key generation**:
   (a) Choose a large prime p and a generator g in the multiplicative $\mathbb{F}_p$.
   (b) Alice selects a secret key a such that $1 \leq a \leq p-1$.
   (c) Compute $h = g^a \pmod{p}$.
   (d) The public key is (p, g, h), and the private key is a.
2. **Encryption**:
   (a) Let m be the message to be encrypted, where $0 \leq m \leq p-1$.
   (b) Bob selects a random integer k such that $1 \leq k \leq p-1$.
   (c) Compute $c_1 = g^k \pmod{p}$ and $c_2 = m \cdot h^k \pmod{p}$.
   (d) Send (c_1, c_2) to Alice.
3. **Decryption**:
   (a) (c_1, c_2) the ciphertext.
   (b) Alice Computes $m = c_2 \cdot (c_1^a)^{-1} \pmod{p}$. This value is the same as plaintext m.

17.5.1 Example of ElGamal Cryptosystem

1. **Key generation**:
   (a) Choose a large prime $p = 467$ and a generator $g = 2$ in the multiplicative group $\mathbb{F}_p$.

(b) Alice selects a secret key $a = 153$ such that $1 \leq a \leq p - 1$.

(c) Compute $h = g^a \pmod{p} = 2^{153} \pmod{467} = 63$.

(d) The public key is $(p, g, h) = (467, 2, 63)$, and the private key is $a = 153$.

2. **Encryption**:

(a) Let $m = 123$ be the message to be encrypted, where $0 \leq m \leq p - 1$.

(b) Bob selects a random integer $k = 85$ such that $1 \leq k \leq p - 1$.

(c) Compute:

$$c_1 = g^k \pmod{p} = 2^{85} \pmod{467} = 61$$

$$c_2 = m \cdot h^k \pmod{p} = 123 \cdot 63^{85} \pmod{467} = 123 \cdot 217 \pmod{467} = 85$$

(d) Send $(c_1, c_2) = (61, 85)$ to Alice.

3. **Decryption**:

(a) Received $(c_1, c_2) = (61, 85)$ as the ciphertext.

(b) Alice computes:

$$s = c_1^a \pmod{p} = 61^{153} \pmod{467} = 217$$

(c) Find the inverse of $s \mod p$. Using Extended Euclidean Algorithm, we get:

$$s^{-1} = 217^{-1} \pmod{467} = 127$$

(d) Recover the message m:

$$m = c_2 \cdot s^{-1} \pmod{p} = 85 \cdot 127 \pmod{467} = 123$$

(e) Thus, the decrypted message is $m = 123$, which is the same as the original plaintext.

17.5.2 ElGamal Analysis

The security of the ElGamal encryption algorithm is based on the hardness of the discrete logarithm problem. Given a large prime p and a primitive root g modulo p, and given $h \equiv g^a \pmod{p}$ for some secret key a, it is computationally infeasible to find a from p, g, and h. This is known as the discrete logarithm problem, and it is to be a hard problem.

The security of the ElGamal encryption algorithm also depends on the randomness of the key used for encryption. If the same key k is used to encrypt multiple messages, an attacker can use the ciphertexts to find the secret key a. Therefore, it is important to choose a different key k for each message that is encrypted.

In addition, the security of the ElGamal encryption algorithm can be enhanced by using a large prime p and a large secret key a. The security of the algorithm also depends on the security of the random number generator used to generate the secret key a and the encryption key k. If the random number generator is predictable, an attacker may be able to predict the secret key a or the encryption key k and break the encryption.

In general, the security of the ElGamal encryption algorithm is considered to be strong, and it is widely used in practice. However, it is important to use appropriate key sizes and follow best practices for key management to maintain the security of the algorithm.

17.6 RSA Cryptosystem

Public and private keys are both used in RSA algorithm. The public key, which is used to convert communications from plaintext to ciphertext, can be known and released to anybody. However, only the accompanying private key may be used to decode communications that have been encrypted using this particular public key. The RSA algorithm's key generation procedure, which has a high level of complexity compared to other cryptosystem methods, is what makes it so safe and dependable today.

The following are the steps involved in the RSA algorithm:

Step 1: Key Generation

1. Alice selects two distinct large prime numbers p and q.
2. Compute $n = p \cdot q$.
3. Compute Euler's Totient function $\varphi(n) = (p-1)(q-1)$
4. Choose an encryption exponent e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$.
5. Compute the decryption exponent d such that:

$$d \cdot e \equiv 1 \mod \varphi(n)$$

6. Alice's public key: (e, n).
7. Alice's private key: (d, n).

Step 2: Encryption (Bob to Alice)

1. Bob obtains Alice's public key (e, n).
2. Represent the message M as an integer where $M < n$.
3. Compute the ciphertext C:

$$C = M^e \mod n$$

4. Bob sends C to Alice.

Step 3: Decryption (Alice's Side)

1. Upon receiving C, Alice uses her private key (d, n).
2. Recover the original message M:

$$M = C^d \mod n$$

17.6.1 Example of RSA Public Key Cryptosystem

Step 1: Key Generation

1. Choose two prime numbers: $p = 11, q = 13$.
2. Compute $n = p \cdot q = 11 \times 13 = 143$.
3. Compute Euler's Totient:

$$\varphi(n) = (p-1)(q-1) = 10 \times 12 = 120$$

4. Choose public exponent $e = 7$ such that $\gcd(7, 120) = 1$.
5. Compute the private key d such that:

$$d \cdot e \equiv 1 \mod 120$$

   The solution is $d = 103$.

6. Public key: $(e = 7, n = 143)$.
7. Private key: $(d = 103, n = 143)$.

Step 2: Encryption (Bob to Alice)

1. Message $M = 9$.
2. Compute ciphertext C:

$$C = M^e \mod n = 9^7 \mod 143 = 48$$

3. Bob sends $C = 48$ to Alice.

Step 3: Decryption (Alice's side)

1. Compute the original message M:

$$M = C^d \mod n = 48^{103} \mod 143 = 9$$

2. Alice recovers the message $M = 9$.

Result: The decrypted message is $M = 9$.

17.6.2 RSA Analysis

The security of the RSA cryptosystem is fundamentally based on the computational difficulty of factorizing the product $n = p \cdot q$, where p and q are large prime numbers. As of now, no efficient algorithm exists for factorizing large numbers in polynomial time, making RSA secure if appropriately large primes are used. The hardness of this factorization problem is what prevents adversaries from deriving the private key from the public key. The larger the primes, the more secure the system becomes, as the time required to factorize n grows exponentially with its size.

A crucial aspect of RSA security is the choice of key size. Commonly used key sizes are 2048 bits or greater, which are currently considered secure against brute-force attacks. Increasing the key size exponentially increases the difficulty of factorization and ensures that the encryption remains safe even with advances in computational power. However, larger key sizes also demand more processing power for encryption and decryption, creating a trade-off between security and performance.

Despite its robustness, RSA is vulnerable to several potential attacks. One of the primary threats is brute-force attacks, which involve attempting all possible keys until the correct one is found. This method, however, becomes infeasible with sufficiently large key sizes. Mathematical attacks focus on exploiting weaknesses in the number theory behind RSA, primarily through factorization techniques. Factoring attacks directly target the difficulty of breaking n into p and q. Another mathematical approach, discrete logarithm attacks, is impractical for RSA due to the nature of modular arithmetic.