Chapter 1

BASIC KNOWLEDGE

1.1. ENCRYPTION

1.1.1. Coding history

The history of encryption spans thousands of years, with developments from simple systems to the complex algorithms used today. Encryption is not only a tool to protect information, but also the foundation of safety and security in the digital world.

1.1.1.1. The Ancient Period: The Beginning of Encryption

During this period, simple encryption systems were created for the purpose of securing information during battles and military communications.

Atbash cipher (circa 6th century BC): Used in ancient Hebrew texts, Atbash is a form of permutation encryption, in which the first letter is replaced by the last letter, the second letter is replaced by the last letter. replaced by the penultimate letter, and so on.

Polybius Cipher (2nd century BC): Invented by Greek philosopher Polybius, this cipher uses a 5x5 square table to convert letters into numerical coordinates.

Caesar Cipher (circa 58-50 BC): Julius Caesar used a simple substitution encryption method, called the Caesar Code, to encrypt military messages. This is an alphabetic shift encoding with a fixed number of shifts. For example, with a 3-position shift, “A” becomes “D”.

1.1.1.2. The Middle Ages and Renaissance: Development of diverse encryption

During this period, encryption began to become more complex, with the emergence of multitable encryption systems and letter pair encryption.

Vigenère Cipher (1553): Blaise de Vigenère, a French diplomat, developed a multitable encryption system. This encoding uses a keyword to adjust the displacement of characters in the message. With this method, each letter in the plaintext is encrypted with a different shift table, making decryption without a key very difficult.

Playfair Cipher (1854): Charles Wheatstone developed this encryption system in which pairs of letters in the plaintext are encrypted instead of individual letters, increasing complexity and security. information.

Morse cipher (1837): a method used in telecommunications to encode text characters as a standard sequence of two different signal intervals, known as *dots* and *dash (dash).*

1.1.1.3. The 20th Century and the Two World Wars: Great advances in encryption

With the development of technology and computers, encryption became an important element in warfare and national security systems.

Enigma encryption (1920s): The Enigma machine was used by the Nazis during World War II to encrypt military information. Each message is encoded using rotating rotors that can change into millions of different configurations. Alan Turing's team at Bletchley Park (UK) developed methods to successfully decrypt this encryption system, greatly contributing to the outcome of the war. Cracking the Enigma code helped shorten the war by about 2 years and saved millions of lives.

Claude Shannon (1949): American mathematician Claude Shannon, in his study "A Mathematical Theory of Cryptography", gave the definition of encryption according to mathematical principles. His research became the foundation for modern cryptographic theory.

Huffman coding (1952) is an encryption algorithm used to compress data. It is based on the frequency table of the characters to be encoded to build a set of binary codes as prefix codes for those characters so that the capacity (number of bits) after encoding is smallest.

1.1.1.4. The birth of modern computers and cryptography (1970 - present)

With the development of computers, encryption systems have become increasingly complex and sophisticated, from symmetric encryption to asymmetric encryption.

DES Encryption (1977): Data Encryption Standard (DES) was developed by IBM and approved by NIST (US National Institute of Standards and Technology). This is a symmetric encryption standard with a 56-bit key, once considered unbreakable. DES uses a 56-bit key, generating 72,057,594,037,927,936 (2^56) possible keys.

RSA Encryption (1977): Rivest, Shamir and Adleman developed the RSA asymmetric encryption algorithm, using public and private keys. This is the foundation of public encryption and is widely used in secure transactions. RSA uses very large prime numbers, up to hundreds of bits. Currently, 2048-bit RSA keys are commonly used for high security.

AES encryption (2001): Advanced Encryption Standard (AES) was developed to replace DES. AES uses 128, 192, and 256-bit key sizes. This is one of the fastest, most secure and efficient encryption methods available today. AES 256-bit has 2^256 possible keys, equivalent to a number that is almost impossible to crack by computational means. modern.

1.1.1.5. Evolution of quantum cryptography and blockchain (21st Century)

Cryptography is not only growing in the traditional digital space but is also expanding into new areas such as quantum computing and blockchain technology.

Shor's Algorithm (1994): Peter Shor develops a quantum algorithm that can factor prime numbers very quickly, threatening the security of RSA-based encryption systems[1] and ECC. A quantum computer using the Shor algorithm can crack 2048-bit RSA in just a few hours, while today's supercomputers take millions of years.

[1] RSA, ECC: xxxx

Blockchain and Bitcoin (2009): Bitcoin and blockchain use advanced cryptographic concepts, including cryptographic hash functions and digital signatures to secure transactions. This is one of the biggest applications of cryptography in the financial system. The SHA-256 hash function, used in the Bitcoin blockchain, produces 2^256 possible results.

1.1.2. Concept of encoding and decoding

Encryption is the process of converting information from a readable form (plaintext) to a form that cannot be read without a decryption tool or key (ciphertext). The goal of encryption is to protect sensitive information, ensuring that only those with access rights can read and understand the information.

Decoding is the reverse process, that is, converting information from encrypted form (ciphertext) back to original form (plaintext). This process can only be performed if the decryption key or corresponding algorithm is known exactly.

Encryption relies on two basic components:

Encryption algorithm (Cipher): This is a method or set of rules used to convert from plaintext to ciphertext. Some popular algorithms include AES, RSA, and DES.

Encryption key (Key): This is a special value that both encryption and decryption processes rely on. Without this key, the decryption process cannot be performed.

There are two types of encryption: symmetric encryption and asymmetric encryption:

Symmetric Encryption: Both encryption and decryption use the same secret key. This algorithm is faster but requires keeping the key secure when shared with the recipient (e.g. AES, DES).

Asymmetric Encryption: Uses a pair of keys, including a public key for encryption and a private key for decryption. This method is more secure but slower (e.g. RSA, ECC).

Encryption has many practical applications

Personal and financial information security: Encryption is widely used in banking and e-commerce transactions to protect credit card data and personal information.

Email and message security: Encryption technology such as PGP (Pretty Good Privacy) is used to protect email content from being read.

Virtual private network (VPN): Uses encryption to protect network connections from hackers or cyber spies.

1.1.3. Classic encryption

Classical Cryptography is a field of encryption that appeared before the advent of computers. These encryption techniques are based on character substitution or permutation and primarily serve military, political, and diplomatic purposes. This section covers important knowledge about classical cryptography, including well-known methods and basic concepts.

1.1.3.1 Substitution Cipher

Substitution encoding is a classic encoding technique in which each character in the plaintext is replaced by another character. Common types of alternative encoding include:

Mother Caesar (Caesar Cipher): One of the oldest encryption methods, used by Julius Caesar to encrypt military messages. Caesar cipher is a type of substitution encoding in which each character in the plaintext is shifted a fixed number of places in the alphabet.

If p is the original character and p is the number of shifted positions, then the encoded character c is calculated according to the formula: $c = (p+k) \bmod 26$. To decode: $p = (c-k) \bmod 26$.

For example: With shift 3, the letter "A" becomes "D", "B" becomes "E", and so on.

Atbash Cipher (Atbash Cipher): This is a form of reverse substitution encoding, in which the first letter of the alphabet is replaced by the last letter, the second letter is replaced by the penultimate letter, and so on.

For example, "A" will become "Z", "B" will become "Y", and "C" will become "X".

Polyalphabetic Cipher: This encoding uses various substitution tables to encode characters in different positions. Each character in the plaintext is encoded in a different table, making it more difficult to crack.

1.1.3.2 Transposition Cipher

Permutation encoding changes the order of characters in the original text instead of replacing them with other characters. This technique is often used in conjunction with substitution encoding to increase complexity.

Scytale Encryption: Scytale was a Spartan encryption method that used a stick (scytale) to write messages on a strip of paper wrapped around the stick. When the paper is removed, the text becomes unreadable unless it is re-wound on a stick of the same diameter.

Column Encoding (Columnar Transposition): In columnar encoding, the plaintext is written into rows of a table with a predetermined number of columns. The characters are then read in top-to-bottom order across the columns instead of left to right.

For example: Original text “HELLOWORLD” with key HACK

The data is represented in a 4-column table according to the length of the HACK

H	A	C	K
H	AND	L	L
THE	IN	THE	R
L	D		

Arranging the columns alphabetically according to the key we get

A	C	H	K

AND	L	H	L
IN	THE	THE	R
D		L	

Now we get the data in each column with the encoded result "EWDLOHOLLR"

1.1.3.3 Vigenère Cipher (Vigenère Cipher)

This is a form of multi-table encoding that uses a keyword to decide how to encode each character in the original text. Each letter in the keyword is used to define a translation table for the corresponding characters in the original text. This makes Vigenère encryption much more difficult to crack than simple substitution encryption.

If the ith character in the original text is p_i and the ith character in the keyword is k_i, then the encoding character c_i is calculated as follows: $c_i = (p_i + k_i)$ mod 26. To decode: $p_i = (c_i - k_i)$ against 26

Example: With the original text "ATTACKATDAWN" with the key "LEMON":

Plaintext: A T T A C K A T D A W N

Key: L E M O N L E M O N L E

Ciphertext: L X F O P V E F R N H R

1.1.3.4. Mã hóa Playfair (Playfair Cipher)

Invented by Charles Wheatstone in 1854, Playfair cipher is a pairwise encoding system. Each pair of letters is encoded based on a 5x5 square board containing the letters of the alphabet (removing the shared Q or I/J).

Encoding rule: With a pair of letters:

If two letters are on the same row, replace each letter with the letter to the right of it.

If two letters are in the same column, replace each letter with the letter below it.

If two letters form a rectangle, replace each letter with the letter in the same row in the remaining corner of the rectangle.

For example:

Playfair encoding table (does not contain "Q"):

P L A Y F

I/J B C D E

G H K M N

O R S T U

V W X Y Z

Encoding the pair "HI" is in a different row and different column, so we take the rectangular corner in the same row "H" and replace it with "G" and "I" replace it with "B".

Encoding the pair "SC" is in the same column, so take the character to move down 1 row, "S" becomes "X" and "C" becomes "K".

Encode the pair "EB" in the same row, so shift right one row "E" to "I/J" and "B" to "C".

1.1.3.5 Hill Cipher

Hill encryption was invented by Lester Hill in 1929, using linear mathematics and matrix algebra to encode blocks of characters. This method uses a square matrix to encode characters in the original text.

Formula: Given a character block of n characters, Hill cipher uses an n×n key matrix to perform matrix multiplication with the plaintext block. Each letter is converted to a number from 0 to 25 (A=0, B=1, C=2, D=3, E=4, F=5, G=6, H = 7 ... … Z= 25), then apply matrix multiplication to calculate the ciphertext.

Decryption is performed by multiplying by the inverse of the key matrix.

For example: With the original text "HOME" and a 2x2 key matrix of:

$$(3\ 3\ 2\ 5\ )$$

Hill cipher will convert "HOME" into the numeric string “HO” (7, 14) and “ME” (10, 4) multiply modulo 26 by the key matrix and then convert the result into cipher text

With “HO” review $(3\ 3\ 2\ 5\ ) * (7\ 14\ ) = (63\quad 84)\ against\ 26\ = (11\ \ 6)$ corresponding to “LG”

With “ME” review $(3\ 3\ 2\ 5\ ) * (10\ 4\ ) = (42\ 40)\ against\ 26\ = (16\ \ 14)$ corresponding to “QO”

So “HOME” will be encoded as “LGQO”.

1.1.3.6. Affine Encryption (Affine Cipher)

This is a form of substitution encryption based on the affine transformation in mathematics. Each character in the plaintext is encoded using a linear function of the form: c=(a·p+b) against 26.

In which: p is the position of the character in the alphabet, a and b are the parameters of the linear function (with a must be coprime to 26) c is the encoding character.

Classical encryption is an important foundation of the field of cryptography. Although these methods are currently not secure enough against modern decryption technologies, they still have great historical value. Classical encryption systems were a stepping stone to the development of more complex encryption techniques, such as symmetric and asymmetric encryption, used in today's digital world.

1.1.4. Symmetric Encryption

Symmetric encryption is one of the oldest and most basic encryption methods, where the same secret key is used to encrypt and decrypt data. This key must be kept secret between the two communicating parties to ensure safety. Symmetric encryption is often used for fast data transmission due to the high speed of the algorithms. Main characteristics of symmetric encryption:

Use a shared secret key: Both parties (sender and receiver) must share the same secret key.

High speed: Symmetric encryption algorithms are generally faster than asymmetric encryption algorithms.

Key distribution problem: Securing and distributing keys between parties is a major challenge in symmetric encryption.

Symmetric encryption is often used in systems that need high speed and large data volumes, such as disk data encryption, network security protocols (VPN, SSL/TLS).

1.1.4.1. Some ciphers are symmetric

a. Data Encryption Standard (DES)

DES was developed by IBM in the 1970s and approved as an encryption standard by NIST (US National Institute of Standards and Technology) in 1977. DES uses a 56-bit key (actually the key is 64-bit). bits, but 8 bits are used for integrity checking). DES is a block cipher with a 64-bit block size, using a block encryption algorithm with a chain of transformations (Feistel network). With the increase in computing power, DES's 56-bit key became insecure. DES has been successfully attacked many times through brute force attacks.

For example: Suppose you want to encrypt the message "HELLO", DES will split the text into 64-bit blocks and encrypt each block with a 56-bit key.

b. Triple DES (3DES)

Triple DES was developed to increase the security of DES by applying the DES algorithm three times with two or three different keys. 3DES uses 112-bit (using two keys) or 168-bit (using three keys) keys. 3DES encrypts data using three passes of DES: encryption, decryption, and encryption again (Encrypt-Decrypt-Encrypt). Although 3DES improves security, it is still considered slow and has short key lengths compared to modern algorithms.

For example: Instead of encrypting "HELLO" once with DES, 3DES will perform three encryptions with different keys to increase security.

c. Advanced Encryption Standard (AES)

AES was developed in the late 1990s to replace DES and 3DES, and was approved by NIST in 2001 after a global competition to find a secure and efficient encryption algorithm. This algorithm supports three key lengths: 128-bit, 192-bit, and 256-bit. AES is a block encryption algorithm with a fixed block size of 128-bit, using a Substitution-Permutation Network. AES has the advantage of being safe, fast and widely used in modern security applications such as SSL/TLS, WPA2 (Wi-Fi), and disk data encryption.

Example: Encrypt the text "HELLO WORLD" using AES with a 128-bit key: Split the text into blocks: AES requires 128-bit blocks as input, so the text "HELLO WORLD" will be converted into binary blocks, which are then split into appropriate blocks (including adding padding if necessary).

AES will use a 128-bit key to encrypt each block, going through 10 rounds of transformation including the steps of replacing bytes (SubBytes), permuting rows (ShiftRows), mixing columns (MixColumns), and adding keys (AddRoundKey). .

Result: The result is an encrypted binary data string that can only be decrypted using the original 128-bit secret key.

d. Blowfish

Blowfish was developed by Bruce Schneier in 1993 as a free alternative to DES. The key of this algorithm can vary from 32-bit to 448-bit, allowing flexibility in security. Blowfish is a block cipher with a 64-bit block size, using a Feistel network with 16 loops. Blowfish is very fast on 32-bit systems, can change key length, and is widely used in applications such as password security.

For example, you can use a 128-bit key to encrypt "HELLO" into a 64-bit block.

e. Twofish

Twofish was developed by the designers of Blowfish as an enhanced version in 1998 and was one of the finalists for the AES competition. Twofish supports 128-bit, 192-bit, and 256-bit keys. Twofish is a block cipher with a 128-bit block size, featuring a Feistel structure with 16 loops. It also uses XOR operations, permutations, and transformations on data bytes. Twofish is highly flexible and secure, designed for speed and safety across both hardware and software.

For example, when encrypting the string "HELLO" with a 256-bit key in Twofish, the message is divided into 128-bit blocks and encrypted through 16 rounds of transformation.

f. RC4 (Rivest Cipher 4)

RC4 is a stream cipher developed by Ron Rivest in 1987. Instead of block encryption, it encrypts each byte of data sequentially. The RC4 algorithm supports keys from 40-bit to 2048-bit. RC4 generates a random keystream and XORs it with each byte in the plaintext to create the ciphertext. Although RC4 is very fast and widely used (in SSL, WEP), it has been found to have many security vulnerabilities, leading to its replacement in many applications.

For example, if you encrypt the string "HELLO" using RC4, each character will be XORed with a byte of the keystream to create the ciphertext.

g. Serpent

Serpent was developed in 1998 by Anderson, Biham, and Knudsen as a candidate for AES. The Serpent algorithm supports 128-bit, 192-bit, and 256-bit keys. Serpent uses an alternative permutation network structure with 32 transformation rings, with the goal of prioritizing safety over speed. Serpent was designed to defend against all known attacks at the time of development and is considered more secure than AES, albeit slower.

For example, when encrypting "HELLO" with a 128-bit key in Serpent, the message is divided into 128-bit blocks and goes through 32 rounds of transformation.

1.1.4.2. Block Cipher Modes of Operation

Block cipher algorithms such as DES, AES, and Blowfish operate on fixed-sized data blocks (for example, AES encrypts 128-bit blocks). However, real data is often not evenly divided into fixed-sized blocks. To solve this problem and handle variable-length data blocks, as well as increase security, modes of operation have been developed. The common modes of operation of block ciphers are presented below:

ECB (Electronic Codebook): In ECB mode, each data block will be encrypted independently. If there are two identical blocks in the plaintext, they will be encrypted as two identical blocks in the ciphertext.

ECB is simple and fast since each block can be encrypted in parallel. However, ECB is not secure when the data has repeating patterns, because the same blocks will produce the same ciphertext, revealing the data structure. For example, images encoded with ECB retain the patterns of the original image.

ECB is rarely used in modern security systems due to its security vulnerabilities.

For example: When encoding an image file with identical color areas, ECB will create identical encryption blocks for these areas, revealing the image pattern.

CBC (Cipher Block Chaining): Each block of data will be XORed with the previous encrypted block before being encrypted. The first block is XORed with a random initialization value called Initialization Vector (IV).

CBC solves the repetition problem of ECB by ensuring that the same blocks will generate different cryptographic blocks, as long as the IV is different. It also creates randomness even when the original text has repeating patterns. CBC does not support parallel encryption, as each block depends on the previous block. This makes CBC slower when processing a lot of data at once.

CBC is widely used in security protocols such as SSL/TLS and IPsec.

For example: When encrypting the string "HELLO WORLD" with CBC, each block will be XORed with the previous encrypted block, creating different encrypted blocks even if the original string has the same parts.

CFB (Cipher Feedback): CFB is a stream mode based on block coding. Instead of encrypting the entire block, CFB encrypts a portion of data in chunks (for example, 8-bit or 64-bit), then XORs this chunk with the plaintext to form the ciphertext.

CFB allows encoding and decoding of variable-sized data segments, and is therefore suitable for continuous data communication (streaming). However, like CBC, CFB does not support parallel encoding.

CFB is commonly used for data stream encryption in real-time communication and transport networks.

For example: When encoding a character string using CFB, the system will process each small segment of data and XOR it with the previously encrypted text.

OFB (Output Feedback): OFB is also a stream cipher mode, but instead of using the output of the previous block, it generates a random key sequence from the IV's encryption. This key string will be XORed with each piece of plaintext data to create encrypted text.

OFB enables parallel processing and protects data from transmission errors, as errors in one block do not affect subsequent blocks. But OFB requires that IVs be kept secret and not reused, because reuse of IVs can lead to security vulnerabilities.

OFB is suitable for applications that require robust protection against communication errors, such as satellite communications encryption.

For example, when encrypting a data string with OFB, each block will be XORed with a portion of the random key string to create the cipher text.

CTR (Counter Mode): In CTR, each block will be XORed with the output of a counter that has been encrypted using the block cipher algorithm. The counter increments with each block, generating a unique keychain for each block.

CTR enables parallel encoding and decoding, which significantly increases performance, especially on multi-core hardware. It also requires no padding, which simplifies encoding of variable-length data. Like OFB, CTR has the disadvantage of requiring counter values not to be reused, otherwise it will create a security hole.

CTR is widely used in security protocols such as IPsec and large data encryption applications.

For example: When encrypting a large file with CTR, the counter will be encrypted to produce different values for each block and XOR with the original text.

1.1.5. Asymmetric Encryption

Asymmetric encryption, also known as public key encryption, is an encryption system that uses two different but linked keys: a public key and a private key.). The public key is used to encrypt data and the private key is used to decrypt data. The special thing is that data encrypted with a public key can only be decrypted with the corresponding private key and vice versa.

Public key: Widely shared and used to encrypt data.

Secret key: Keep secret and used to decrypt data.

Some typical asymmetric encryption algorithms

a. RSA (Rivest-Shamir-Adleman)

RSA is based on the difficulty of factoring a large number into prime factors. The public key and private key are generated based on pairs of large prime numbers.

This is a popular algorithm and has been widely used in systems such as SSL/TLS, secure email (PGP), and digital signatures. But RSA requires a large key length to ensure security, consuming computational resources.

For example, during a secure web session, the server's RSA public key is used to encrypt data transmitted between the server and the browser.

b. DSA (Digital Signature Algorithm)

DSA is specifically designed to create and verify digital signatures. DSA does not directly encrypt data, but only creates digital signatures to authenticate the integrity and origin of the information.

The advantage of DSA is optimization for signing and verifying digital signatures. Its disadvantage is that it cannot be used to encrypt data.

For example, DSA is commonly used in systems like GPG to authenticate emails and documents.

c. ECC (Elliptic Curve Cryptography)

ECC relies on the mathematical properties of elliptic curves to generate public and private keys. ECC provides the same level of security as RSA with a much smaller key size.

ECC has better performance than RSA, especially on devices with limited resources (e.g., mobile phones, IoT). However, it is necessary to choose the correct elliptic curve parameters to ensure security.

For example, ECC is used in modern security protocols such as HTTPS, VPN, and blockchain (for example, Bitcoin uses ECC to manage wallet keys).

d. ElGamal

ElGamal is an encryption algorithm based on the difficulty of the discrete logarithm problem. It provides both data encryption and digital signature generation features.

ElGamal has the advantage of good security with large keys, capable of resisting many types of attacks, but the encrypted text size is larger than RSA and ECC.

For example, ElGamal is often used in systems that require encryption and digital signatures at the same time.

e. Paillier

Paillier is a homomorphic encryption algorithm, which allows performing mathematical operations on encrypted text without decrypting it.

Paillier is applicable to systems that require secure computation on encrypted data, such as cloud storage services. However, it has poorer performance than other algorithms in common applications.

For example, Paillier is often used in applications that require computation on confidential data without decryption, such as financial data processing.

1.1.4.3. Some practical examples

SSL/TLS Protocol: During an SSL/TLS session, asymmetric encryption (usually RSA or ECC) is used to exchange the initial symmetric key. Then, symmetric encryption (usually AES) is used to encrypt subsequent data to optimize performance.

Email digital signature (PGP/GPG): The sender uses his private key to create a digital signature, and the recipient uses the sender's public key to verify that the email has not been altered and is from sender.

Blockchain: In blockchain like Bitcoin, ECC is used to manage wallet keys. Users create a public wallet address to receive funds, and only they with the corresponding private key can spend those funds.

1.2. HASH FUNCTION

1.2.1. Concept and role of hash function in security

A hash function is a mathematical function that converts input data of any size (e.g. text, files, numbers) into a fixed-length string of characters, called is a hash value or hash. This value uniquely represents the original data and changes completely if just a bit of information in the input data is changed.

The hash function has several key properties

Deterministic: The same input always produces the same hash value.

Fixed size: Whatever the input length, the output of the hash function always has a fixed size.

Fast calculation: The hash function must calculate the hash value quickly even with large amounts of data.

Collision-resistant: It is difficult or impossible to find two different data with the same hash value.

One-way: It is difficult or impossible to recreate the original data from the hash value.

Avalanche Effect: A small change in the input data will cause a significant change in the hash value.

The role of hash functions in security

Hash functions play an important role in many areas of information security, helping to ensure integrity and protect data from attacks. Here are some prominent roles:

Ensuring data integrity: Hash functions are used to check data integrity, ensuring that data has not been altered during transmission or storage. For example, when downloading a file from the internet, a hash value of the file (usually SHA-256 or MD5) is provided so that the user can compare the hash value of the downloaded file with a published hash value. dad. If these values match, it proves the file has not been changed.

Digital Signatures: In the process of creating a digital signature, a hash function is used to create a value that uniquely represents the document or message. This hash value is then encrypted with the sender's private key to create a digital signature. The recipient uses the sender's public key to check the signature and compare the hash value with the original document to authenticate its integrity and origin.

Password Encryption: In login systems, instead of storing the user's original password, the system often stores the hash value of the password. When a user enters a password, the system hashes the password and compares the hash value with the saved value. This helps protect passwords from being exposed when data is hacked. Techniques such as salting (adding a random string to the password before hashing) help increase security, making dictionary attacks or Rainbow Table attacks (pre-hashed tables) more difficult.

Blockchain: In blockchain systems like Bitcoin, hash functions are used to link blocks together and ensure the integrity of the chain. Each block contains a hash of the previous block, making changing data in one block requiring changing all the blocks after it, which is nearly impossible. The mining process in blockchain is also based on finding hash values that match certain requirements, through very complex calculations.

Authenticate data in communication protocol

Hash functions are used in security protocols such as SSL/TLS, IPsec, HMAC (Hashed Message Authentication Code) to ensure that messages are not altered during transmission and to authenticate the authenticity of the parties. join.

HMAC is a technique that combines a secret key and a hash function, helping to protect data from modification attacks.

Virus and malware detection: Virus detection software often uses hash functions to compare files with known virus samples. If the hash value of a file matches the hash value of a virus sample, the software will warn about the existence of malware in the system.

1.2.2. Distinguish between encryption and hash function

Both encryption and hashing are security techniques used to protect information, but they have completely different natures and goals.

a. Encryption

Encryption is the process of converting original data (plaintext) into encrypted data (ciphertext) so that unauthorized people cannot read the data. This process is reversible: the recipient can decrypt the data to recover the original information using a secret key (in symmetric encryption) or a pair of public and private keys (in symmetric encryption). in asymmetric encryption).

The goal of encryption is to ensure the confidentiality of data by hiding its content from unauthorized parties.

Encryption uses two types of keys: Symmetric encryption: Uses the same key for both encryption and decryption (e.g. AES, DES). And asymmetric encryption: Using a pair of public and private keys, the public key is used for encryption and the private key is used for decryption (eg: RSA, ECC).

Practical example: HTTPS protocol: When you visit a website via HTTPS, data between the browser and web server is encrypted using algorithms such as AES (symmetric) and RSA (asymmetric) to protect protects content from attackers during transmission.

Secure Email (PGP): When sending a secure email using PGP, the data is encrypted with the recipient's public key, only their private key can decrypt and read the email content.

Reversal process: Can be decoded to retrieve the original information (plaintext).

b. Hash Function

A hash function is a mathematical function that converts input data of any length into a fixed string of characters (hash value) without decoding back to the original data. The hash function is designed to be irreversible (one-way), which means that the original data cannot be recovered from the hash value.

The goal of a hash function is to ensure data integrity by checking whether the data has been altered or not. The hash function is not used to secure content, but to compare and verify data.

The hash function does not require a key during calculation. The hash value depends entirely on the content of the data.

Some practical examples:

Check file integrity: When downloading a file from the internet, the website may provide a hash value (e.g. SHA-256) so the user can check if the downloaded file has been altered. or damaged during loading.

Password: In login systems, instead of storing the password in plain text, the system stores the hash value of the password. When a user enters a password, the system will hash the password and compare it with the saved hash value. This helps protect passwords from being exposed when the database is compromised.

Reverse process: Cannot decrypt, can only compare hash values to check data integrity.

c. Compare encryption and hash functions

Encryption in online banking: When you make transactions via online banking, your financial data is encrypted using symmetric (like AES) and asymmetric (like RSA) algorithms to ensure that only the bank can read this sensitive information.

Hash function in checking file integrity: When downloading a software update from the developer's website, they provide the SHA-256 hash value of the file. Once downloaded, you can compare the hash value of the file on your computer with the hash value provided. If it matches, the file was not changed or corrupted during the download.

Encryption and hashing combine in password security: When you log in to a system, your password is encrypted during transmission from the browser to the server for security (for example, over HTTPS). Once it reaches the server, the password is hashed (usually with SHA-256) and that hash value is compared to the hash value stored in the database to confirm the correctness of the password without needing to know it. root password.

As such, encryption and hashing are both important security tools, but they serve different purposes. Encryption is used to protect the confidentiality of information during transmission or storage, while hashing is used to check data integrity and protect the system from unwanted changes.

1.2.3. Popular hash functions

Hash functions are an important component in security and cryptographic systems. It helps protect data integrity, authentication, and security in many applications. This section covers popular hash functions in more detail

1.2.3.1 MD5 (Message Digest Algorithm 5)

The MD5 algorithm developed by Ronald Rivest in 1991 is the successor to MD4, with the goal of improving security and performance. Before the security vulnerability was discovered, MD5 was widely used in verifying data integrity.

The concept of MD5 is designed to convert an input of any length into a fixed 128-bit hash value. MD5 uses a process consisting of four loops that process data through a series of addition and bit shifting operations.

The MD5 algorithm divides the input information into 512-bit blocks, then performs a series of block XOR, addition, and shift operations to produce the final hash value. This procedure includes several loops with secret operations.

For example: The string "hello" has MD5 hash code: 5d41402abc4b2a76b9719d911017c592

MD5 is fast to calculate and easy to deploy and use in many applications.

The disadvantage of MD5 is that it is susceptible to collision attacks, where two different messages can give the same hash value. This reduces its safety. Another drawback is that due to security vulnerabilities, MD5 is no longer safe for security applications such as password authentication or digital signatures.

MD 5 is commonly applied:

+ Check file integrity: MD5 is used to confirm that the file has not been altered when transmitted over the internet.

+ Password encryption: Previously, MD5 was used to hash passwords in systems, but has now been replaced by more powerful algorithms.

1.2.3.2. SHA-1 (Secure Hash Algorithm 1)

The SHA-1 algorithm was developed by the NSA (US National Security Agency) in 1993. The SHA-1 algorithm is widely used in many security applications, including SSL/TLS certificates, but research Research shows that SHA-1 is no longer secure.

The idea of SHA-1 is to generate a 160-bit hash value from input of any length. The hashing process includes XOR operations, bit shifts, and additions in loops to produce the hash value.

The SHA-1 algorithm uses 5 blocks of 32-bit data in 80 iterations. Each loop applies an XOR operation, an addition operation, and a logic operation.

Example: SHA-1 hash "hello" string: 2ef7bde608ce5404e97d5f042f95f89f1c232871

SHA-1 is widely used in digital security and authentication protocols. Ensures information integrity in applications such as digital signatures and SSL certificates.

However, SHA-1 has some disadvantages:

+ Collision vulnerability: Since 2005, researchers have shown that SHA-1 is vulnerable to collision attacks.

+ Deprecation recommendation: NIST has advised against using SHA-1 for security applications because of its vulnerability.

Apply:

+ Digital signatures and SSL/TLS certificates: SHA-1 is used in the process of creating digital signatures for SSL certificates.

+ Message authentication: Before being replaced, SHA-1 was used to authenticate message integrity in security protocols.

1.2.3.3. SHA-2 (Secure Hash Algorithm 2)

The SHA-2 algorithm developed by the NSA in 2001 is an improved version of SHA-1, designed to overcome the vulnerabilities of SHA-1.

The idea of SHA-2 is that it is a family of hash functions that includes versions SHA-224, SHA-256, SHA-384, and SHA-512, each with a different hash length. The hashing process in SHA-2 uses addition, XOR, and shift operations to produce a secure hash value.

The SHA-2 algorithm applies a more robust series of logical operations than SHA-1, with hash lengths varying depending on the version used.

For example: String: "hello" has a SHA-256 hash of

2cf24dba5fb0a30e26e83b2ac5b9e29e1b169e7e9e1c8e4e58c96d9e2b9bda94

SHA-2 is more secure than SHA-1 and MD5, recommended for modern security applications such as blockchain, SSL/TLS. High security, resistant to collision attacks.

SHA-2 speed may be slower than MD5 and SHA-1, but in return it provides greater security.

Apply:

+ Blockchain: SHA-256 is used in blockchains like Bitcoin to secure transactions.

+ Digital signature: SHA-2 is used in SSL/TLS certificates and digital signatures to ensure data integrity and authenticity.

1.2.3.4. SHA-3 (Secure Hash Algorithm 3)

The SHA-3 algorithm was developed by NIST in 2015 to replace SHA-2 in case SHA-2 is broken in the future. The SHA-3 algorithm uses a different structure than SHA-2 (Keccak), providing greater security.

The concept of SHA-3 uses a completely different structure called Keccak, which is based on the sponge method, different from the Merkle–Damgård based methods of SHA-1 and SHA-2.

Keccak's algorithm divides information into blocks and uses addition, XOR, and bit shifting operations to create the hash value.

For example: String: "hello" has SHA-3-256 hash code b94d27b9934d3e08a52e52d7da7dabfad3a9c404b85e347506f906420dce2086

The advantage of SHA-3 is that it is highly secure, no collision errors are detected. The Keccak structure is more robust than SHA-2 in protecting against potential attacks.

Although powerful, SHA-3 has not been as widely adopted as SHA-2.

Apply:

+ High security applications: SHA-3 can be used in security applications that require extremely high security.

+ Future Blockchain: SHA-3 can be used to replace SHA-2 in future blockchain systems.

1.2.3.5. BLAKE2

The BLAKE2 algorithm was developed by researchers Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn in 2012. Development: BLAKE2 was designed as an alternative to MD5 and SHA-2, with High performance and better security.

BLAKE2 is a fast and secure hash function that can replace MD5 and SHA-2 in many applications that require high speed and security.

BLAKE2 uses a Merkle tree structure and has very high performance compared to SHA-2.

Example: BLAKE2b-256 hash string "hello": f1d2d2f924e986ac86fdf7b2a5a7f52d

Pros: Fast speed, high performance, Strong security and easy to deploy.

Disadvantages: Not as widely available as SHA-2, although it is powerful and fast.

Apply:

+ File systems: BLAKE2 is used in many file systems and applications that need fast hash calculations.

+ Security software: BLAKE2 is used in security and data verification software.

Each hash function has unique characteristics and is suitable for different needs. MD5 and SHA-1 have gradually become obsolete due to security vulnerabilities, while SHA-2 and

SHA-3 are the main choices in modern security applications. BLAKE2 is a powerful new choice with fast performance.

1.2.4. Some applications of hash functions

Hash functions have many applications in security and information systems, especially in ensuring data integrity, protecting passwords, and supporting secure transactions. Here are some common applications of hash functions:

1.2.4.1. Software and File Integrity Checking

Hash functions are widely used to check the integrity of software or files downloaded from the internet, to ensure that the files have not been altered or hacked.

- Download software from websites: Websites that provide software or files (such as operating system ISOs, software, documents) often publish the hash value (MD5, SHA-1 or SHA-256) of the file. Users can download the software and recalculate the hash value of the file after downloading, if the hash value matches, the file has not been changed.

- Security software: Tools like Tripwire check for changes to files in the system and alert users when unwanted changes occur. These changes may be the result of malware or illegal manipulation.

1.2.4.2. Password Management Systems

The hash function helps protect the user's password in the login system. When a user creates a password, the system does not store the original password but stores the hash value of that password. When a user logs in, the system hashes the password and compares it with the stored hash value.

To increase security, when hashing passwords, systems add a random string (salt) to the password before calculating the hash value, helping to prevent dictionary attacks and hacking attacks. oil stains (rainbow table attacks). For example, bcrypt and Argon2 use salts to protect user passwords.

1.2.4.3. Blockchain and Cryptocurrency (Cryptocurrency and Blockchain)

Hash functions are an essential component in blockchain systems such as Bitcoin and Ethereum, helping to protect the integrity of transaction data and create blocks in the chain. Each block contains the hash value of the previous block and the new transaction, forming an immutable chain.

- Bitcoin: SHA-256 is used in protecting transactions and blocks in the Bitcoin blockchain. Each block in the chain contains the hash value of the previous block, creating a linked chain that cannot be changed.

- Ethereum: Ethereum uses SHA-3 to hash transactions and blocks in its blockchain network.

1.2.4.4. Digital Signatures

The hash function is an important part of a digital signature, helping to authenticate the integrity of data and verify the identity of the signer. The process of creating a digital signature includes hashing the data and encrypting the hash value with the signer's private key.

- SSL/TLS Certificate: Hash functions such as SHA-256 are used to hash digital certificate data during the digital signature creation and verification process, helping to ensure the integrity and origin authentication of SSL certificates /TLS.

- Electronic signature: Organizations, banks, and electronic services use digital signatures to sign contracts, documents, or online transactions. For example, digital signatures in electronic banking transactions help confirm the legitimacy of the transaction.

1.2.4.5. Database Integrity Checking

In database management systems (DBMS), hash functions can be used to check for changes to records in the database. This helps determine if there has been unauthorized access or unwanted data changes in the database.

- Detect fraud in banking transactions: Banks can use hash functions to track financial transactions and check data integrity.

- Change control: Database management tools can use hash functions to validate that there have been no unauthorized changes in important database records.

1.2.4.6. Message Authentication

A hash function combined with a secret key can generate a message authentication code (MAC) to ensure message integrity and authenticity in electronic transactions or network communications.

- HMAC (Hash-based Message Authentication Code): HMAC uses a hash function (such as SHA-256) combined with a secret key to create an authentication code for the message. This helps protect messages from being altered during transmission. HMAC is used in security protocols such as SSL/TLS, IPsec and SSH.

- API and Web Services: In API transactions or server-to-server communication, HMAC is used to protect data integrity and prevent man-in-the-middle attacks.

1.2.4.7. Distributed File Systems

Hash functions help ensure data integrity in distributed storage systems such as IPFS (InterPlanetary File System) and Ceph. Files in these systems are often hashed and stored with the hash value as an index, allowing for data validation and quick searching.

- IPFS: Data in IPFS is divided into blocks, each block is assigned with a unique hash value. This helps ensure file integrity when stored and shared on distributed networks.

- Ceph: The Ceph distributed storage system uses hash functions to track and check the integrity of files in the system.

1.2.4.8. Network Authentication and Secure Protocols

Hash functions are a key component in authenticating secure protocols such as SSL/TLS and IPsec. They help protect data from alteration and ensure the authenticity of transactions.

- SSL/TLS: Web security protocols such as SSL/TLS use hash functions to authenticate data during the connection between client and server. SHA-256 is one of the popular hash functions in SSL/TLS.

+ IPsec: In virtual private networks (VPNs), IPsec uses HMAC to protect the integrity and authenticity of data packets during transmission over the network.

1.2.4.9. Fraud Detection and Anti-Attack

Hash functions can help detect fraud and attacks in security systems. For example, in financial transaction systems, hash functions can be used to examine transactions for unusual activity.

- Online payment systems: The system can use hash functions to check for illegal or fraudulent transactions in financial transactions.

- Denial of service (DDoS) attack detection: Security tools can use hashing to monitor access patterns and detect distributed denial of service (DDoS) attacks or other network.

Hash functions have a wide range of applications in modern IT and security fields, from personal data protection, message authentication, to security in blockchain transactions and network systems. Understanding and correctly applying hash functions in each context is important to ensure system security and integrity..

1.3. DIGITAL SIGNATURE

1.3.1. Concept of digital signature

Digital Signature is a security technology used to authenticate and protect the integrity of information in the electronic environment. It is like a handwritten signature, but used in electronic transactions and documents. Digital signatures help ensure that a message or document has not been altered after signing and confirm that the signer is who they claim to be.

Digital signatures not only help authenticate the identity of the sender, but also prove that the data has not been modified after the signature was created. To perform digital signature, use cryptographic algorithms, especially asymmetric encryption, including public and private keys.

1.3.2. Operating principle of digital signature

Digital Signature operates based on the principle of asymmetric cryptography and hash function. This process has the main purpose of authenticating the identity of the signer and ensuring that the content of the document or message has not been changed after signing.

Digital signature includes two main steps: creating a digital signature and checking the digital signature. Below is a detailed description of its operating principle:

a. Create digital signature

The process of creating a signer's digital signature can be described through the following steps:

Step 1: Generate the hash value of the message

The sender (or signer) calculates the hash value of the message or document to be signed (for example, an electronic contract). To calculate the hash value, the signer will use a hash function (e.g. SHA-256). This hash value is a fixed-length string of numbers and characters that represents the content of the message.

Step 2: Encrypt the hash value with a private key

After calculating the hash value, the signer will use his private key to encrypt this hash value. This encryption process creates a digital signature. This digital signature will be attached to the message sent to the recipient. Only the owner of the private key can encrypt and create a digital signature, ensuring that the signature is unique and cannot be forged.

Step 3: Send message and digital signature

The signer sends both the message and the digital signature to the recipient. At this point, the recipient will be able to use the digital signature and the message to verify the validity of the message and the signer.

b. Check digital signature

When the recipient receives the message and digital signature, they will perform the following steps to check the validity of the signature:

Step 1: Create hash value of received message

The recipient will recalculate the hash value of the message or document they receive using the same hash function that the signer used (e.g., SHA-256). This creates a new hash value of the message.

Step 2: Decrypt the digital signature using the public key

The recipient will use the signer's public key to decrypt the digital signature they receive. Decryption returns the original hash value that the signer encrypted.

Step 3: Compare hash values

The recipient will compare the hash value they calculated (from the message) with the hash value they obtained from decrypting the digital signature. If these two hashes match, it means the message has not been altered and the signature is valid. If there is no match, the message has been changed or the signature is invalid.

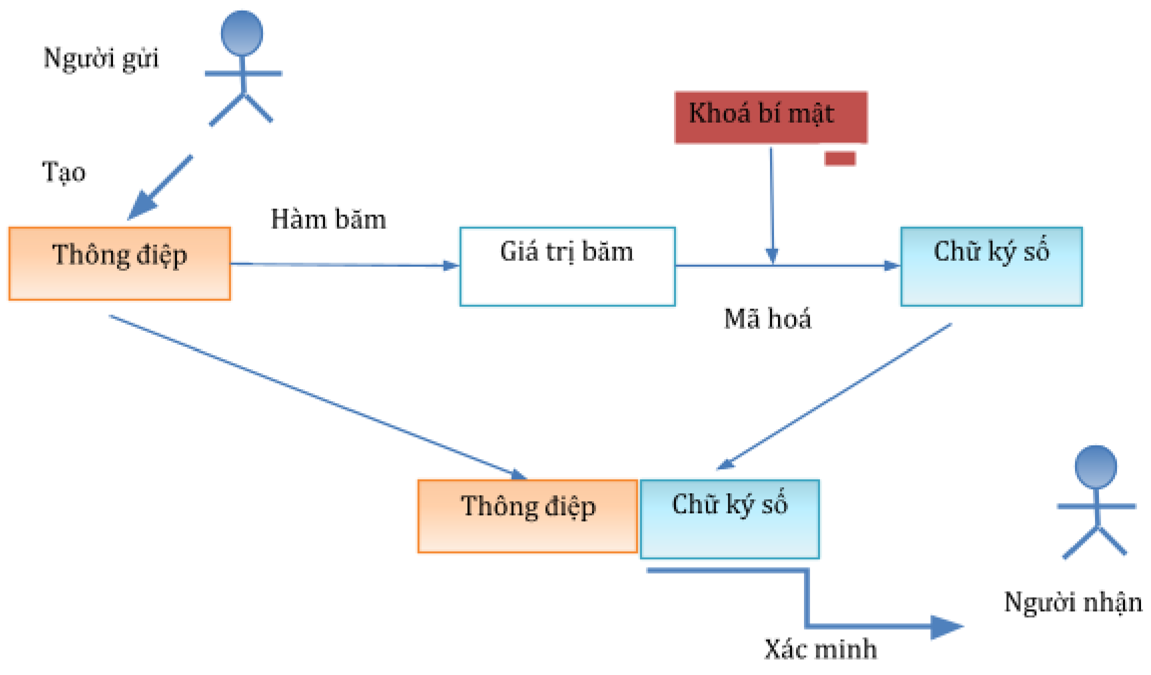

Figure 1: Diagram illustrating the digital signature process

1.3.3. The role of digital signatures

Digital signatures are used for digital documents, indicating that the entire document has been signed by the person signing. And other people can verify this. Digital signatures are similar to regular signatures, ensuring the document content is reliable, accurate, unchanged during transmission and indicates who created the document.

However, a digital signature is different from a regular signature, because it depends on the text. Digital signatures will change according to the document, while regular signatures will not change at all.

Digital signatures are used to provide owner authentication, data integrity, and non-repudiation in a variety of fields.

1.3.4. Components of digital signature

Digital Signature includes the following main components, helping to authenticate the identity of the signer and protect the integrity of the message. These components are created and used during the signing and verification of digital signatures:

a. Private Key

The private key is part of an asymmetric key pair that belongs only to the signer and is used to encrypt the hash value of the message.

The private key helps the signer create a digital signature by encrypting the hash value of the message. This key is strictly guarded and must not be shared with anyone.

The private key must be kept absolutely secret. If exposed, an attacker can forge the signer's digital signature.

b. Public Key

The public key is the remainder of the asymmetric key pair and is shared publicly with all recipients.

The public key is used to decrypt the digital signature received by the recipient. By using the public key, the recipient can check whether the digital signature is valid and whether the message has been altered.

The public key can be distributed widely, as long as the private key remains secure.

c. Message or document (Message/Document)

This is the message or document that the signer wants to protect and authenticate.

The message is the content that the signer wants to convey. A digital signature ensures that the message remains unchanged after signing and confirms the identity of the signer.

The integrity of the message is protected thanks to the use of a hash function and hash value encryption.

d. Hash Value (Hash Value)

A hash value is the result of using a hash function (e.g., SHA-256) on a message or document. This is the numerical representation of the message, encrypted to create a digital signature.

The hash value is an arithmetic representation of the message, helping to minimize the size of the data and speed up the signing process. It also ensures that any small change in the message will result in a significant change in the hash value.

The hash value is irreversible and cannot be tampered with. This helps ensure message integrity during transmission.

e. Digital Signature

A digital signature is the result of encrypting the hash value of the message with the signer's private key. Used to confirm the validity of the message

Digital signatures ensure the authenticity of the signer and protect the integrity of the message. This signature will be sent with the message so the recipient can check its validity.

Digital signatures cannot be forged, because only the signer's private key can create the digital signature. If the message is altered, the digital signature will no longer be valid.

f. Hash Function

A hash function is a one-way encryption algorithm used to convert a message into a fixed-length hash value. The algorithm used to generate the hash value of the message, protecting its integrity

The hash function ensures message integrity. If the message is altered, the hash value will change and the recipient will recognize this change when checking the digital signature.

Popular hash functions such as SHA-256, SHA-3 have strong properties such as collision-resistant and one-way function, helping to protect information from being tampered with.

Digital signatures are a powerful tool in authenticating and protecting information in the electronic environment, helping to ensure data integrity and verify the identity of the signer. The components of a digital signature, including the private key, public key, hash value, and hash function, together create a strong security system for electronic transactions and documents.

1.3.5. Detailed process for creating and verifying digital signatures

1.3.5.1. Create digital signature

The process of creating a digital signature includes three main steps: creating a hash function, encrypting the hash function, and concatenating information. These steps are described in detail as follows:

Step 1: Create hash function (Hashing)

Input: Data to be signed (message or digital document).

Implementation: Apply a cryptographic hash function to the input data. Commonly used hash functions include SHA-256, SHA-3, or SHA-512.

Result: A hash value of fixed size, called the "message digest". This hash value is unique to the content of the message, i.e. if the message changes, the hash value will also change.

For example: Suppose the message to be signed is "Hello, World!".

Applying the SHA-256 hash function, the result will be the hash value: a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b59f1b4d89013dd31.

Step 2: Encrypt the hash function

Input: Hashed message from step 1 and signer's private key.

Implementation: Use an asymmetric encryption algorithm (such as RSA or ECDSA) to encrypt the hash value. The private key is used to perform encryption.

Result: Digital signature, which is an encrypted value based on the message hash and the signer's private key.

Example: Let's say we use the RSA algorithm and the signer's private key. The hash value from step 1 is encrypted with the private key, creating a digital signature.

If a private key is used, the digital signature after encryption can be: 93844c24b8f502bc32ab8a... (this value will vary depending on the key and encryption algorithm).

Step 3: Pair information

Input: Original message and digital signature.

Implementation: The digital signature is combined with the original message to form a signed data package.

Result: The data package contains the message and digital signature, which can be sent or stored.

The message "Hello, World!" concatenated with the digital signature 93844c24b8f502bc32ab8a..., forming a signed data packet.

1.3.5.2. Verify digital signature

The digital signature verification process involves verifying whether the signature is accurate and has not been tampered with. The verification process includes the following steps:

Step 1: Regenerate the hash from the message

Input: The original message received from the signed data packet.

Implementation: Apply the same hash function used when creating the signature to recreate the hash value from the message.

Result: New hash value from received message.

For example: The receiver receives the message "Hello, World!" from the signed data packet and apply the SHA-256 hash function to the message. The result is the hash value:

a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b59f1b4d89013dd31.

Step 2: Decode the digital signature

Input: Digital signature and public key of the signer.

Implementation: Use the public key and asymmetric encryption algorithm to decrypt the digital signature, obtain the hash value that was encrypted during the signature creation process.

Result: The hash value was previously encrypted.

For example: The recipient uses the signer's public key to decrypt the digital signature 93844c24b8f502bc32ab8a.... The decryption process will obtain the original hash value (encrypted from the original message):

a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b59f1b4d89013dd31.

Step 3: Compare two hash values

Input: Hash value from received message (step 2.1) and decrypted hash value from signature (step 2.2).

Implementation: Compare these two hash values.

Result: If the two hash values are the same, the digital signature is verified as valid; otherwise, the signature is invalid or the message has been altered.

1.3.5.3. Specific examples

Suppose Alice wants to digitally sign a document to send to Bob:

Alice:

+ Use the SHA-256 hash function on her document and get the hash value.

+ Use RSA private key to encrypt hash values and create digital signatures.

+ Send documents with digital signature to Bob.

Bob:

+ Receive documents and digital signatures from Alice.

+ Calculate the hash value of the document you receive.

+ Use Alice's public key to decrypt the digital signature and obtain the original hash value.

+ Compare two hash values. If they are the same, Bob knows that the document has not been altered and the digital signature is valid, confirming Alice is the one who signed.

1.3.6. Digital signature application

Digital signatures have many applications in different fields, thanks to their authentication, security and data integrity features. Here are important applications of digital signatures:

a. Electronic transactions and e-commerce

Digital signatures are used to sign electronic contracts, electronic invoices, and documents related to business transactions. It ensures that the sender and recipient can both verify each other's identities and that the content is not altered after signing.

For example: In e-commerce systems, businesses and customers use digital signatures to sign sales contracts, payments and other legal agreements.

b. Electronic government

Digital signatures are an important component in e-government implementation, helping to authenticate the identities of citizens and government agencies in online transactions.

For example: Citizens can pay taxes, register for public services, or sign administrative documents online without having to go to government agencies.

c. Electronic invoice (e-Invoice)

Digital signatures are used to sign and authenticate the validity of electronic invoices, ensuring that the invoice sent cannot be modified and the receiving party can trust the authenticity of the invoice.

For example: In accounting systems, businesses sign electronic invoices with digital signatures to send to customers and tax authorities.

d. Secure emails and messages

Digital signatures help secure important emails and messages by ensuring that the content is not altered and the sender's identity is authenticated.

For example: In internal business communications, digital signatures ensure the confidentiality and integrity of sensitive information transmitted via email.

e. Securities and financial transactions

Digital signatures are used in securities and financial transactions to authenticate the identities of participating parties and ensure that signed financial documents are accurate and cannot be altered.

For example: Online stock trading transactions often require digital signatures to ensure the safety and legality of the transaction.

f. Online banking

In online banking, digital signatures help authenticate financial transactions, such as fund transfers, loans, and account openings. This helps minimize fraud risks and protect customers' rights.

For example: When customers make a large transaction or sign a loan contract online, they can use a digital signature for confirmation.

g. Sign a contract remotely

Digital signatures allow parties to sign contracts remotely without having to meet in person. This helps save time and costs during the contract signing process.

For example: In multinational companies, partners in different countries can use digital signatures to sign contracts without meeting in person.

h. Software authentication

Software developers use digital signatures to sign applications, to confirm that the software is trustworthy and has not been changed since its release.

For example, when downloading an application from a trusted developer, a digital signature on the software helps users confirm that it has not been modified or injected with malicious code.

i. Blockchain and cryptocurrency

Digital signatures are an integral part of blockchain and cryptocurrency transactions, helping to authenticate ownership and integrity of transactions.

For example, in the Bitcoin and Ethereum blockchain systems, digital signatures help confirm that a transaction was made by the true owner of the cryptocurrency wallet address.

j. Electronic health

In the medical field, digital signatures help authenticate patient records and electronic medical documents, ensuring the security and integrity of medical information.

For example, doctors can use digital signatures to sign medical records and electronic prescriptions, helping patients easily store and manage medical records online.

1.3.7. Benefits and limitations of digital signatures

Digital signatures bring many benefits in securing and authenticating online transactions, especially in the fields of e-commerce, e-government and finance. However, the implementation of digital signatures also faces some limitations related to cost, complexity, and legal issues. Effective use of digital signatures requires careful consideration of these benefits and limitations.

1.3.7.1. Benefits of digital signatures

a. High security

Digital signatures use asymmetric encryption, ensuring high security thanks to the separation between public and private keys. Only the holder of the private key can create a digital signature, while anyone can use the public key for verification. Digital signatures minimize the risk of tampering or modifying information during transmission.

b Identity authentication

Digital signatures help authenticate the identity of the signer, ensuring that the document or transaction was made by the correct entity. Digital signatures help ensure reliability in electronic transactions, preventing fraud and counterfeiting.

c. Data integrity

The hash function in the digital signature creation process helps detect any changes to the information after signing. If data is modified after signing, the digital signature will become invalid. It ensures that information is not altered during transmission or storage.

d. Legality

Digital signatures are legal in many countries, considered legal evidence equivalent to hand signatures in many regulations and laws related to electronic transactions. Digital signatures enable signers to sign contracts and legal agreements remotely, increasing efficiency and saving time.

e. Save costs and time

Using digital signatures eliminates the need to print and deliver physical documents, as well as manual signing and verification. Digital signatures help speed up the processing of transactions and reduce operating costs related to paperwork and delivery.

g. Easily integrated with digital systems

Digital signatures can be easily integrated into document management systems, electronic transactions, and e-commerce platforms. It helps improve operational efficiency and automate many business processes.

1.3.7.1. Limitations of digital signatures

a. Initial setup costs

Implementing digital signatures requires establishing a public key infrastructure (PKI), including the issuance and management of digital certificates from certification authorities (CA - Certificate Authority). Therefore, the cost of setting up and maintaining a PKI system can be high, especially for small and medium-sized organizations.

b. Depends on the certification authority (CA)

Digital signatures require trust in a certification authority (CA) to issue and validate a user's digital certificate. Therefore, if the certification authority is compromised or stops working, the security and authenticity of the digital signature may be affected.

c. Difficult to use for people unfamiliar with technology

Users who are unfamiliar with concepts related to digital signatures, digital certificates, and encryption may have difficulty creating, managing, and using digital signatures. Therefore, training and technical support are needed for users to understand how to effectively use digital signatures.

d. Term of digital certificate

Digital certificates have a validity period, and after expiration, users need to renew or re-register for a new digital certificate. Therefore, managing the term and renewal of digital certificates can cause inconvenience, especially if the renewal process is not done in a timely manner.

e. International legal issues

Although digital signatures are recognized in many countries, there still exist differences in legal regulations between countries regarding the recognition of digital signatures, especially in international transactions. Thus, a limitation is that transactions between parties in different countries may encounter legal barriers related to the recognition and use of digital signatures.

g. Risk of losing private key

Digital signatures rely on keeping the user's private key secret. If the private key is lost or stolen, the person's digital signature can be forged. Therefore, users need to manage private keys carefully to avoid loss or misuse.

1.4. DECENTRALIZED SYSTEM

1.4.1. Decentralized System concept (Decentralized System)

A decentralized system is a system where control and decision-making is not centralized in a single entity, but is divided among many individual nodes or components. In this system, there is no central power unit with full control, instead the nodes in the system have equivalent roles and work independently or together to achieve a common goal.

Decentralized systems are often applied in areas such as blockchain, computer networks, decentralized autonomous organizations (DAOs), and decentralized governance. In this system, the distribution of control and data helps reduce centralization risks, increase security, and improve transparency.

1.4.2. Characteristics of decentralized systems

No single center of control: Decision-making and data authority do not depend on a single entity.

Self-managed and distributed: Nodes in the system operate and manage themselves without central intervention.

Enhanced fault tolerance: If one or more nodes fail, the system can continue to operate, increasing durability.

Transparent and not easily tampered with: Thanks to decentralization, every change in the system can be audited and tracked.

1.4.3. Structure and model of decentralized system

Decentralized systems have a different structure than centralized systems, because control and resources are not concentrated in a single entity but are distributed among many nodes. Main components in the structure of a decentralized system:

a. Nodes

Nodes are basic components in a decentralized system, which can be computers, servers, or personal devices, each node is capable of processing and storing data independently. Nodes have equal rights and responsibilities.

Nodes perform tasks such as verification, transaction processing, data storage, and ensuring system integrity. Some systems may have specialized buttons such as:

- Validators: In the blockchain, these nodes participate in the transaction validation process.

- Storage Nodes: Store data and redistribute it when required, for example in peer-to-peer (P2P) network systems.

b. Consensus Protocol

In a decentralized system, nodes must agree on the state of the system without intervention from a central entity. To do this, a consensus protocol is used to achieve agreement between nodes regarding actions and changes in the system.

For example: Consensus protocols[2] Popular in blockchain include:

- Proof of Work (PoW): Nodes solve cryptographic problems to confirm transactions (used in Bitcoin).

- Proof of Stake (PoS): Nodes contribute an amount of digital assets to confirm transactions (used in Ethereum 2.0).

- Delegated Proof of Stake (DPoS): Nodes vote to select representatives to validate transactions.

c. Distributed ledger[3] (Distributed Ledger)

The distributed ledger is where all the transactions or events that have been confirmed by the nodes in the system are stored. Each node has a copy of this ledger and has the right to audit transactions at any time.

The distributed ledger ensures transparency and data integrity, as every change is recorded and distributed to all nodes.

For example, Blockchain is a popular form of distributed ledger in which each block contains transactions and is linked to the previous block.

[2] This concept is presented in detail in chapter 4 of this textbook
[3] This concept is presented in detail in chapter 2 of this textbook

d. Peer-to-Peer Communication mechanism between nodes

Nodes in a decentralized system communicate directly with each other without going through a central server. Communication between nodes typically takes place via peer-to-peer (P2P) network protocols.

The communication mechanism helps ensure that messages and data can be exchanged quickly between nodes. This helps the system continue to operate even when some nodes are disconnected.

For example: In P2P networks like BitTorrent, computers share files directly with each other instead of through a central server.

e. Distributed Resources

Resources in a decentralized system are not concentrated in one place but are distributed among nodes. This can include data, bandwidth, processing capacity or digital assets.

Distributed resources enhance the system's fault tolerance and ensure higher availability compared to a centralized system.

For example: Decentralized storage systems like IPFS distribute data across multiple nodes instead of storing all files in a single location.

f. Smart contracts[4](Smart Contracts) (optional)

Smart contracts are programs that automate agreements or transactions when conditions are met. They are often deployed on decentralized platforms such as blockchain.

Smart contracts allow to automate processes and carry out transactions without the need for intermediaries.

For example, on the Ethereum blockchain, smart contracts can automatically execute transactions when predetermined conditions are met.

g. Security and Cryptography

Decentralized systems rely on cryptographic techniques to ensure that information and transactions are protected from tampering and intrusion. This includes data encryption, digital signatures, and hash functions to protect integrity.

Cryptography helps authenticate identities, protect privacy, and ensure the integrity of data in the system.

1.4.4. Advantages of decentralized structure and decentralized system

[4] The concept of smart contracts is presented in detail in chapter xx of this textbook

With the above characteristics and decentralized structure, the model has the following advantages:

- Fault and attack resistant: There is no single point of failure, the system can still operate even if some nodes are compromised.

- Enhanced security: Using strong and distributed cryptography helps prevent tampering and ensures data security.

- High transparency: All transactions and changes are auditable by nodes, making the system transparent.

Along with that, the decentralized system brings benefits such as:

- High security and privacy: Because there is no single centralized point for attack, decentralized systems are more difficult to breach or misuse data than centralized systems.

- Anti-censorship: There is no central agency that can interfere or censor information in the system.

- Increased transparency and trust: All transactions or changes in the system can be recorded and verified by all nodes.

- Reduce the risk of system failure: Decentralized systems often have higher fault tolerance, because they do not depend on a single point.

1.4.5. Applications of decentralized systems

Decentralized systems have many applications in different fields thanks to their security, transparency, high fault tolerance, and ability to eliminate dependence on intermediaries. Below are common applications of decentralized systems:

1. Blockchain and cryptocurrency

Blockchain is a type of decentralized system in which transactions are stored in blocks and linked together into chains. There is no central authority that controls the entire system, but instead nodes jointly verify and maintain the ledger. For example:

- Bitcoin: A decentralized cryptocurrency where transactions are verified through a Proof of Work (PoW) consensus mechanism and there are no intermediary banks or financial institutions.

- Ethereum: Blockchain platform that supports smart contracts, allowing for automatic decentralized transactions without the need for intermediaries.

2. Decentralized Autonomous Organization (DAO - Decentralized Autonomous Organization)

DAOs are organizations that operate on their own without a central administrator. The DAO's rules and operating processes are encoded in smart contracts, and DAO members have the right to vote on issues. For example:

- MakerDAO: An organization that allows users to create stablecoin DAI through smart contracts without the need for banks or financial institutions.

- Aragon: A platform that makes it easy to create and manage decentralized organizations on the blockchain.

3. Decentralized Finance (DeFi - Decentralized Finance)

DeFi is a financial ecosystem without the need for traditional intermediaries such as banks and financial companies. Transactions such as borrowing, lending, buying and selling, insurance, and many other financial services are performed through smart contracts. For example:

- Uniswap: A decentralized exchange (DEX) that allows users to buy and sell cryptocurrencies without intermediaries.

- Aave: Decentralized lending platform that allows users to borrow and lend cryptocurrency without going through a bank.

4. Peer-to-peer network (P2P - Peer-to-Peer)

A P2P network is a decentralized network where devices can connect and share resources directly with each other without going through a central server. For example:

- BitTorrent: A peer-to-peer file sharing protocol that allows users to share and download files without the need for a centralized server.

- IPFS (InterPlanetary File System): Decentralized file storage and sharing system, helping to improve data access and security.

5. Decentralized communication

Decentralized media platforms do not rely on intermediary service providers, helping to increase privacy and avoid censorship. For example:

- Matrix: A decentralized open source communication protocol that allows communication systems to connect with each other.

- Signal: Although not yet completely decentralized, Signal uses end-to-end encryption and is evolving towards increased decentralization.

6. Decentralized social network

Decentralized social networks allow users to fully control their content and personal data without relying on intermediary companies or service providers. This helps avoid censorship and better protect privacy. For example:

- Mastodon: Decentralized social network where users can create their own servers to manage and control personal data.

- Steemit: A decentralized social media platform running on the Steem blockchain, where users are rewarded in cryptocurrency for contributing content.

7. Decentralized data storage

Decentralized data storage system helps distribute data across many nodes, eliminating dependence on centralized storage services such as Google Drive, Amazon S3. This provides security and ensures that data is not susceptible to censorship or loss. For example:

- Storj: Decentralized data storage platform that uses blockchain and P2P networks to securely store data.

- Filecoin: Decentralized storage network that allows users to rent storage space from others globally, while ensuring data is safe and uncensored.

8. Decentralized administration

Decentralized systems can be applied to governance models, where decisions are made based on the consensus of the community or organization, without depending on a central authority or leader. For example:

- Liquid Democracy: Decentralized governance model that allows citizens or members of an organization to vote directly or authorize others to vote on their behalf.

- Tezos: A self-governing blockchain where users can vote on protocol changes or improvements without outside intervention.

9. Decentralized health system

Decentralized health systems can improve the storage and sharing of patient data between health organizations without relying on a central authority. This helps secure medical information and allows quick access to patient records. For example:

- MedRec: A project that uses blockchain to store and manage medical records, allowing patients to control and share their medical data.

10. Decentralized Internet of Things (IoT).

Decentralized IoT systems use blockchain or P2P networks to connect and manage devices without central servers. This helps increase security and minimize single points of failure in the network. For example:

- IOTA: A decentralized blockchain platform specifically designed for IoT networks, allowing devices to securely communicate and exchange data without the need for intermediaries.

The application of decentralized systems is expanding strongly in many fields from finance, healthcare, data storage to administration. With high security, transparency and eliminating dependence on intermediaries, decentralized systems not only help reduce costs but also increase user control and system sustainability.

1.5. DISTRIBUTED SYSTEM

1.5.1. Distributed system concept

A distributed system is a collection of independent computers that combine together to form a unified system, operating as a single entity. In this system, components can be located in different locations, and they communicate with each other over the network to perform common tasks. The goal of distributed systems is to provide high performance, availability, and fault tolerance while operating on many separate computers.

1.5.2. Characteristics of distributed systems

Independence: Distributed system components can operate independently and do not need to be in the same location.

Concurrency: Processes or components in a distributed system can operate concurrently, performing tasks in parallel.

Scalability: Distributed systems are easily scalable by adding resources (computers or hardware) without disrupting the system.

Fault tolerance: Distributed systems have good fault tolerance, meaning that if one or a few components fail, the system can still operate normally or with slightly reduced performance.

1.5.3. Distributed system models

In distributed systems, there are many different models for organizing and managing system components. Each model will suit different system goals and requirements such as performance, availability, security, scalability, and complexity. Below are some popular models in distributed systems:

1. Client-Server Model (Client-Server)

The client-server model is one of the simplest and most popular models in distributed systems. In this model, there are two main types of nodes:

- Client: Clients are nodes that send requests to the server and receive responses from the server. Clients are typically terminals such as web browsers or user applications.

- Server: Servers are nodes that provide services or resources to client computers. The server receives requests from the client, processes them, and returns the results.

The advantage of the model is that it is easy to deploy, easy to maintain, and easy to control because there is a central server. Its disadvantages are that the server can become overloaded if too many clients simultaneously request the service, and this model may not be fault tolerant if the central server crashes.

For example: Traditional web application model, where the client (browser) sends an HTTP request to the web server and receives a web page back.

2. Peer-to-Peer - P2P Model

In the Peer-to-Peer model, all nodes in the system have equal rights and responsibilities. There is no central server, and each node can be both a client and a server, sharing resources and services directly with other nodes.

The advantages of P2P are that there is no single point of failure, it is easy to scale, and resources are distributed evenly among nodes. P2P has the disadvantage that system management and security are more complicated since there is no central server, and nodes may not always be available or reliable.

For example: File sharing networks like BitTorrent, where users upload and download files from other computers without the need for a central server.

3. Multi-tier model

The multi-tier model (also known as the three-tier model in web applications) divides the distributed system into multiple tiers to divide specific tasks, making the system more flexible and easier to maintain.

Common tiers in a multi-tier model include:

- User interface layer (Presentation Layer): This is the layer where users interact, such as a web browser or mobile application.

- Business Logic Layer: This layer contains business processing logic and processes for handling requests from users.

- Database layer (Data Layer): This layer manages the database, where data is stored and retrieved.

This model has the advantage of being easy to maintain, expand and divide responsibilities between layers. However, layers can create latency due to the need to go through many steps, especially in a network environment.

For example: A three-tier web application (web application) with a user interface layer (UI), a business processing layer (Business Logic), and a database layer (Database).

4. Distributed File System (DFS) model

In this model, distributed file systems provide an approach to storing and accessing data from multiple distributed nodes that users can access as if that data resides on a single file system.

Files are divided into small blocks of data, distributed across multiple servers, and users can access them through a common interface system.

The advantages of the model are that it is easily scalable, has good fault tolerance, and provides the ability to access files from any node in the system. However, in this model consistent file management and synchronization between nodes can be difficult.

Ví dụ: Hadoop Distributed File System (HDFS), Google File System (GFS).

5. Distributed Database System Model (Distributed Database System)

Distributed database systems divide the database into multiple parts and distribute them across multiple nodes in the system. These nodes can perform data storage and query tasks simultaneously, improving system performance and fault tolerance.

The advantages of this model are that it is easy to expand, data is distributed evenly, and the system has high fault tolerance. However, synchronizing and maintaining consistency between data copies can be complicated.

For example: NoSQL database systems like MongoDB, Cassandra, or distributed SQL database systems like Google Spanner.

6. Cloud Computing Model

The cloud model uses distributed resources from many servers located in different data centers and provides services over the Internet. Users can access these services without having to worry about hardware infrastructure.

The cloud model offers flexible scalability, saves hardware costs, and provides easy access to remote services and resources. This model depends on an Internet connection, and security issues can arise when storing data in the cloud.

For example: Amazon Web Services (AWS), Microsoft Azure, Google Cloud.

7. Distributed Cloud Computing Model (Edge Computing)

In the distributed cloud computing model, resources and services are distributed close to users or edge devices, instead of concentrated in large data centers.

The advantages of the model are reduced latency, bandwidth savings, and fast data processing close to the origin. The model has the disadvantage that data management and security become more complicated.

For example: IoT applications, where data is processed right at sensor devices or network nodes near the user.

1.5.4. Properties of distributed systems

- Transparency:

+ Access transparency: Users do not need to know the physical location of the resource (data, service) in the distributed system, just use it as if it were local.

+ Location transparency: Users do not need to know which machine in the network the resource is located on.

+ Error transparency: The system has the ability to hide errors and recover automatically, helping the system maintain operation.

+ Mobility transparency: Resources or processes can move within the system without affecting users.

- Openness:

Distributed systems can be easily expanded and incorporated with new components without affecting the current system.

Components in the system communicate with each other via standard protocols and easily interact with other systems.

- Synchronous and Asynchronous:

In distributed systems, there can be synchronous systems where components communicate with each other at a specified rate. However, most distributed systems are asynchronous, in that components do not need to wait for each other.

- Decentralization:

There is no central server to manage everything, computers in a distributed system often have equal management and operating rights, helping to reduce centralization risks and increase fault tolerance.

- Fault Tolerance:

Distributed systems have the ability to continue operating even if some components fail. Mechanisms such as backup, redundancy, and failure detection help the system maintain high availability.

- Consistency:

System data and state must be maintained consistently across components in the distributed system. There can be many levels of consistency such as strong consistency, weak consistency or eventual consistency.

- Availability:

The system must always be ready to serve user requests, even if some nodes in the system are not working.

- Scalability:

Distributed systems can scale easily as resource needs increase. This includes scaling both the size of the system and the number of users or workloads.

- Symmetry:

In some distributed systems, components may have equivalent roles, with no clear distinction between server and client.

These properties make distributed systems an effective solution in many situations, from data storage systems, cloud services to Internet of Things (IoT) networks.

1.5.5. Synchronization and integrity management

Synchronization and integrity management are important challenges in distributed systems. These aspects are concerned with ensuring that processes or nodes in the system can operate together without data conflicts or state distortions.

1. Synchronization management in distributed systems:

Synchronization in distributed systems involves ensuring that processes operate in a coordinated manner and avoid conflicts accessing shared resources or data.

Locking Mechanisms: A common method for managing synchronization is to use locking mechanisms such as mutexes, semaphores, or token distribution protocols. This prevents multiple processes from accessing the same resource at the same time, leading to resource contention.

Clock Synchronization: In a distributed system, processes on different machines may have asynchronous system time. Several algorithms are used to synchronize time between nodes such as NTP algorithm (Network Time Protocol), Berkeley algorithm and Lamport Timestamps (to synchronize the order of events).

2. Data integrity in distributed systems:

Data integrity ensures that data is not altered or corrupted as it moves across different nodes of a distributed system.

Hierarchical Synchronization: Some distributed systems use hierarchical methods to ensure data integrity in cases of complete desynchronization between nodes.

Concurrency Control: Parallel control is a technique that ensures that transactions or queries from multiple users are processed in a way that maintains data consistency. For example, Two-Phase Locking (2PL) mechanism is commonly used in distributed database systems.

Consistency Models: In distributed systems, there are many different levels of consistency, from strong consistency to eventual consistency. The choice of consistency model depends on the system requirements, between availability and consistency.

3. Challenges in managing synchronization and integrity:

Network Latency: Because nodes in a distributed system may be located in different geographical locations, network latency and bandwidth can affect data synchronization, leading to discrepancies. time.

Network Partitioning: Network partitioning issues can cause out-of-sync states, affecting data integrity. Algorithms like Paxos and Raft are used to maintain consensus in unreliable network conditions.

4. Synchronization and integrity management solutions:

Distributed Algorithms: Many algorithms have been developed to manage synchronization and integrity such as Chandy-Lamport Snapshot Algorithm, Bully Algorithm, and Paxos.

Replication: Replicating data across multiple nodes helps ensure availability and integrity. However, synchronization between replicas requires robust strategies to avoid consistency problems.

Consensus Mechanisms: Distributed systems can use consensus mechanisms such as Paxos or Raft to ensure that all nodes agree on a specific value before continuing with tasks other.

1.5.6. Applications of distributed systems

Distributed systems have many applications in different fields, from information technology, finance, healthcare to entertainment and scientific research. Below are some common applications of distributed systems:

1. Cloud Computing:

Cloud computing is one of the most popular applications of distributed systems, allowing users to access computing resources and data storage over the Internet.

Platforms such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure are all based on a distributed system model to provide infrastructure services (IaaS), platforms (PaaS), and software (SaaS) for global users.

Services such as cloud storage, cloud computing, and content delivery networks (CDN) all use distributed systems to ensure availability and capacity. extend.

2. Distributed Databases:

Distributed databases allow data to be stored and managed on multiple servers within a network. This helps increase fault tolerance, ease scalability, and reduce access latency.

Systems such as Google Spanner, Amazon DynamoDB, Apache Cassandra, and MongoDB use distributed architectures to manage data efficiently and ensure consistency.

Data partitioning and synchronization mechanisms help ensure data is backed up and synchronized between servers, serving applications that require high performance.

3. Blockchain and cryptocurrency (Cryptocurrency):

Blockchain is a typical application of distributed systems, where data is stored on a network of nodes and maintained through consensus algorithms such as Proof of Work (PoW) or Proof of Stake (PoW). PoS).

Applications like Bitcoin, Ethereum and many other cryptocurrencies operate on a distributed system, without a central managing entity, enhancing security, transparency and decentralization.

Blockchain also has other applications beyond cryptocurrencies, including smart contracts, supply chain management, and digital asset management.

4. Distributed File Systems:

Distributed file systems help store and retrieve data on multiple servers, allowing applications and users to access data remotely easily.

Systems such as Google File System (GFS), Hadoop Distributed File System (HDFS), and Ceph are typical examples of distributed file systems, often used in big data applications (Big Data).

These systems ensure availability and scalability, while keeping data safe and in sync.

5. Social Networking and Communication applications (Social Networking and Communication):

Social networks like Facebook, Twitter, Instagram, and messaging apps like WhatsApp and Telegram all rely on distributed systems to handle billions of requests every day.

The distributed system helps social networks scale, store and manage huge amounts of data, and deliver real-time content to users globally.

Distributed-based communication services also use caching and CDN techniques to improve access speed and stability.

6. Financial and Banking Systems:

Electronic payment systems, online banking, and stock trading systems also use distributed systems to ensure availability, scalability, and security.

For example, Visa, MasterCard, and international stock exchange systems all rely on a distributed model to process millions of transactions per second.

Payment services such as PayPal, Stripe, Square and other currency management systems also use distributed systems to ensure fast and secure transactions.

7. Big Data Analytics (Big Data Analytics):

Distributed systems are the foundation of big data applications, where tools such as Apache Hadoop, Apache Spark, and Google MapReduce are used to process and analyze large volumes of data across distributed computing systems. .

Organizations like Google, Amazon, Facebook use these tools to analyze user behavior, optimize systems and improve service performance.

Big data systems require high scalability and fault tolerance, which distributed systems can provide.

8. IoT (Internet of Things):

An IoT network is a distributed system with millions of devices connecting and sharing data with each other. IoT applications in smart homes, healthcare, transportation, and industry rely on distributed systems to manage and process data from sensors and devices.

Platforms such as AWS IoT, Google Cloud IoT, and Microsoft Azure IoT Hub provide distributed-based services to manage IoT devices and the data generated.

9. Online Gaming:

Massively multiplayer online (MMO) games like World of Warcraft, Fortnite, and League of Legends use distributed systems to manage millions of players worldwide.

The distributed system reduces latency and ensures stable real-time gaming experiences for players regardless of their geographical locations.

The use of distributed servers also makes the game scalable as the number of players increases.

10. Application of artificial intelligence (Artificial Intelligence):

AI and machine learning systems often require large amounts of computing and data resources. Models like TensorFlow and PyTorch can be deployed on distributed systems for training and inference on big data.

AI applications such as image recognition, natural language processing, and complex data analysis can all be deployed on distributed systems to optimize performance.

Thanks to features such as fault tolerance, scalability, and high availability, distributed systems have become an important foundation for many applications in the modern world.

1.6 PEER-PEER NETWORK

1.6.1. Peer-to-peer network concept

Peer-to-Peer (P2P) network is a network model in which nodes in the network all have equal roles. These nodes can act as both a client and a server, meaning they can both receive and provide resources. There is no central management server, all network members can directly exchange data with each other.

The P2P model is often used to share files, computational resources, or connections between multiple devices in a distributed system without the need for an intermediary.

1.6.2. Characteristics of peer-to-peer networks

Distributed (Decentralized): There is no central server to manage the entire system. Members are equal and manage their own resources.

Scalability: When many nodes join the network, the scalability is very high. This increases bandwidth and computing resources without the need for additional central servers.

Fault Tolerance: Peer-to-peer networks often have good fault tolerance, because the failure of a few nodes does not greatly affect the entire system.

Load Balancing: Because there is no central server, data storage and sharing are evenly distributed across nodes, helping to reduce overload on a single server.

Resource Management: Each node can manage its own resources, including bandwidth, memory, and data.

1.6.3. Types of peer-to-peer networks

- Unstructured P2P network (Unstructured P2P):

In an unstructured network, nodes are randomly arranged and there are no fixed rules for connecting nodes to each other. Unstructured P2P networks are easy to set up, no need for complex architecture. However, searching for resources can be inefficient, especially as the number of nodes increases. For example: Gnutella, Napster.

- Structured P2P network (Structured P2P):

In a structured network, nodes are arranged according to a certain rule, often using a distributed hash table (DHT) to manage resources. This type of network is easy to search and access data quickly but is more complicated to set up and maintain than an unstructured network. For example: Chord, Kademlia.

- Hybrid P2P network (Hybrid P2P):

Combination of P2P network and client-server network, where some special nodes act as intermediary servers that help coordinate and manage connections between nodes. This type of network takes advantage of both the P2P and client-server models.

For example: BitTorrent (with trackers acting as intermediary servers).

1.6.4. Mechanism of operation of P2P network

Connection between nodes: Nodes in a P2P network connect directly to each other, forming a distributed network. When a node joins the network, it searches for other nodes to connect to, and can then share or request resources.

Data exchange: Data or resources can be shared between nodes by sending requests to other nodes to search for the desired resource. These requests can be broadcast or sent in a structured way depending on the type of P2P network.

Search and data distribution: One of the important mechanisms in P2P is data search and location. In unstructured P2P networks, search requests are often broadcast throughout the network. In a structured network, the DHT distributed hash table is used to find data quickly.

Swarming: In networks like BitTorrent, large files are divided into small parts and these parts are downloaded from different sources, helping to increase transmission speed.

1.6.5. Technology and protocols in peer-to-peer networks

Distributed Hash Table (DHT): DHT is the main technology for locating and managing data in structured P2P networks. Nodes and resources in the network are mapped to a virtual address space. Each node is responsible for managing only a portion of this space. For example: Kademlia, Chord.

BitTorrent Protocol: BitTorrent is a popular protocol for file sharing in P2P networks. Files are broken into pieces, and users can download pieces from many different people at the same time.

Gnutella: One of the first unstructured P2P protocols. Gnutella uses a broadcast model to search and exchange data between nodes.

FastTrack: Protocol used by famous file sharing networks like Kazaa, uses a number of supernodes to manage and coordinate network traffic.

1.6.6. Peer-to-peer network applications

- File Sharing: The most popular application of P2P networks is file sharing, such as BitTorrent. Users can share music files, videos, software or other documents easily and quickly. Examples of applications in this direction include BitTorrent and eMule.

Media Streaming: P2P networks are also used for video streaming and multimedia streaming applications. Content transmitted via P2P network can reduce the load on the central server and increase user access speed. For example: Joost, Popcorn Time.

- Distributed Computing: P2P networks allow taking advantage of the computing power of many distributed computers to perform complex tasks. For example: SETI@home (search for alien life), Folding@home (research on proteins).

- Blockchain and Cryptocurrency System: Blockchain is an application of a P2P network in which transactions are confirmed and recorded on a distributed ledger without the intervention of an intermediary. For example: Bitcoin, Ethereum.

- Communication Networks: P2P networks are used in messaging and peer-to-peer communication applications, where messages and data are transmitted directly between devices without the need for an intermediary server. Skype systems (previously using P2P), Tox are examples of this type of application.

QUESTIONS AND EXERCISES

1. What is the difference between symmetric encryption and asymmetric encryption?
2. List the symmetric encryption algorithms and explain how they work?
3. Name asymmetric encryption algorithms and explain why they are often used in information security?
4. What is the block cipher mode?
5. What is the concept of padding in encryption?
6. What is the role of the session key in symmetric encryption?
7. What is a hash function?
8. Which property of a hash function is most important?
9. How is a cryptographic hash function different from a regular hash function?
10. What is the difference between encryption and hashing?
11. Why is hash collision a security problem?
12. How does digital signature work?
13. What is the difference between digital signature and electronic signature?
14. What is the role of hash function in digital signature?
15. What are the common algorithms used in digital signatures?
16. How to verify the validity of a digital signature?
17. How do digital signatures help prevent attacks?
18. Write a program to encrypt and decrypt a message using the AES algorithm. Experiment with different operating modes such as ECB, CBC, and GCM.
19. Write a program to create a hybrid encryption system that uses asymmetric encryption to encrypt the session key and symmetric encryption to encrypt the message.
20. Uses the AES algorithm in ECB mode to encrypt a bitmap image. Observe the results and explain why ECB mode should not be used to encode large structured data such as images.
21. Write a simple program to generate a hash function based on a character string (e.g., using the sum of the ASCII codes). Let's experiment with some different input strings

22. Given a given hash string (e.g.: `d2d2d2...`), find an input string that can generate this hash. Try with popular hash functions like MD5, SHA-1, SHA-256 and record the execution time.

23. Try finding two strings that are different but produce the same hash value using a simple self-written hash function. Compare the results with popular cryptographic hash functions such as MD5 or SHA-1.

24. Write a program to check the integrity of a file using a hash function (for example, SHA-256). Let's change some bytes in the file and observe the change in the hash value.

25. Explain the Birthday attack and write a program that simulates this attack principle on a simple hash function, then implement it with a hash function such as MD5 or SHA-1.

26. Write a simple program that uses the RSA algorithm to create a digital signature for a text string and then verify this signature. Experiment with popular cryptographic libraries like `PyCryptodome` or `OpenSSL`.

27. Uses the ECDSA algorithm to create and verify digital signatures. Compare the performance of ECDSA with RSA when signing the same message.

28. Download a digital certificate from a website (for example, via HTTPS) and write a program to extract the public key from this certificate. Use the public key to verify a digital signature.

29. Research how digital signatures are used in smart contracts on the blockchain platform. Write a simple smart contract that uses digital signatures to verify transactions.

30. Perform a hands-on exercise to create and verify a digital signature using both RSA and ECDSA, then compare signature size, signing speed, and verification speed.

REFERENCES

Books:

1. Stallings, William. "Cryptography and Network Security: Principles and Practice" (7th Edition)
2. Schneier, Bruce. "Applied Cryptography: Protocols, Algorithms, and Source Code in C" (2nd Edition)
3. Stinson, Douglas R. "Cryptography: Theory and Practice"
4. Katz, Jonathan; Lindell, Yehuda. "Introduction to Modern Cryptography"
5. Menezes, Alfred; van Oorschot, Paul C.; Vanstone, Scott A. "Handbook of Applied Cryptography" (CRC Press, 1996)
6. Schneier, Bruce. "Applied Cryptography: Protocols, Algorithms, and Source Code in C" (John Wiley & Sons, 2015)

7. Coulouris, George; Dollimore, Jean; Kindberg, Tim; Blair, Gordon. "Distributed Systems: Concepts and Design" (5th Edition)
8. Tanenbaum, Andrew S.; Van Steen, Maarten. "Distributed Systems: Principles and Paradigms" (2nd Edition)
9. Tanenbaum, Andrew S.; Van Steen, Maarten. "Distributed Systems: Principles and Paradigms" (2nd Edition, Pearson Education, 2016)
10. Coulouris, George; Dollimore, Jean; Kindberg, Tim; Blair, Gordon. "Distributed Systems: Concepts and Design" (5th Edition, Addison-Wesley, 2011)
11. National Institute of Standards and Technology (NIST) Special Publications (SPs) & FIPS PUBs:
12. FIPS PUB 197: Advanced Encryption Standard (AES)
13. FIPS PUB 180-4: Secure Hash Standard (SHS)
14. FIPS PUB 186-4: Digital Signature Algorithm (DSA)
15. SP 800-38A: Recommendation for Block Cipher Modes of Operation
16. SP 800-56A: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography

Request for Comments (RFCs):

1. RFC 3447: Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1
2. RFC 1321: MD5 Message-Digest Algorithm

Online Resources:

1. "The Twofish Encryption Algorithm: A 128-Bit Block Cipher" by Bruce Schneier et al.
2. "Serpent: A Proposal for the Advanced Encryption Standard" by Ross Anderson et al.
3. "The RC4 Stream Cipher and Its Variants" by Ronald Rivest
4. J. Daemen and V. Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard"
5. NIST Special Publication 800-38A, "Recommendation for Block Cipher Modes of Operation"
6. "Public Key Infrastructure (PKI) and Digital Signatures," Microsoft Documentation
7. "Digital Signatures," National Institute of Standards and Technology (NIST)
8. "The Salsa20 Family of Stream Ciphers," by Daniel J. Bernstein
9. The Elliptic Curve Cryptography (ECC) Primer, Certicom Research
10. ISO/IEC 10118-3: Information technology — Security techniques — Hash-functions
11. Public Key Infrastructure (PKI) Standards
12. Directive 1999/93/EC of the European Parliament and of the Council on a Community framework for electronic signatures
13. NIST Digital Signature Standard (DSS)
14. Decentralized Finance (DeFi) Pulse
15. The InterPlanetary File System (IPFS) Documentation
16. Chainlink Blog

17. Ethereum Official Website
18. Coursera: "Cloud Computing Specialization" - University of Illinois
19. MIT OpenCourseWare: "Distributed Systems"
20. "What is a Distributed System?" - Martin Kleppmann
21. Leslie Lamport (1978). Time, Clocks, and the Ordering of Events in a Distributed System
22. Eric A. Brewer (2000). Towards Robust Distributed Systems
23. Ian Clarke et al. (2001), Freenet: A distributed anonymous information storage and retrieval system
24. John Buford, Heather Yu, Eng Keong Lua (2009), P2P Networking and Applications
25. Stefan Saroiu, Steven D. Gribble, Henry M. Levy (2003), Measurement and Analysis of Spyware in a University Environment
26. Bram Cohen (2008), The BitTorrent Protocol Specification
27. Leonard Kleinrock (1976), Queueing Systems Volume 2: Computer Applications
28. David P. Anderson (2004), BOINC: A System for Public-Resource Computing and Storage
29. M. Castro, P. Druschel, A-M.