Hash Functions, MACs, and Digital Certificates

Q. What is a hash function?

A **cryptographic hash function** is a **mathematical algorithm** that takes an input (message or data) of any size and produces a fixed-size **hash value** (also called a **digest**), which uniquely represents the input.

It is a **one-way function**:

H(message)=digest

You **cannot reverse** the digest to get the original message.

Purpose:

Hash functions are primarily used for:

- **Data integrity**: To ensure data hasn’t been altered
- **Authentication**: Often used with digital signatures
- **Message verification**: To confirm the original message hasn't changed

---

How It Works:

1. **Sender** computes a **message digest** (MD) using a hash function on the original message:
2. MD = Hash(message)
3. **Sender** sends both the message and the message digest to the **receiver**.
4. **Receiver** receives the message and computes the hash again:
5. MD' = Hash(received_message)
6. **Compare**:
    - If MD == MD', then message is authentic and unaltered.
    - If MD ≠ MD', then the message has been modified.

Properties of a Good Hash Function:

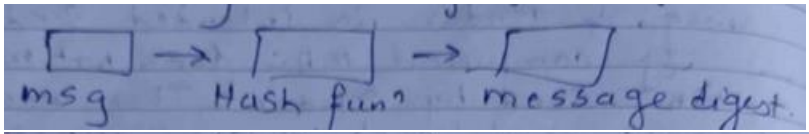
Property	Description
One-way	It should be computationally infeasible to reverse a hash (i.e., get the original message from the hash).
Fixed output size	Regardless of input size, output is of fixed length (e.g., 256 bits for SHA-256).
Deterministic	Same input always produces the same output.
Avalanche effect	A small change in input causes a large, unpredictable change in output.
Collision-resistant	It should be hard to find two different inputs that produce the same output hash.
No secrecy	Hashing is not encryption — no key is involved. Anyone can compute a hash.

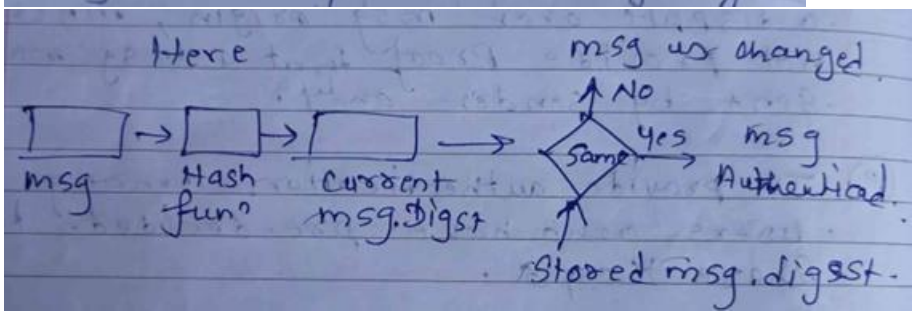
Example:

Let's say:

- **Message**: "I Love Icecream" → Hash → 12345
- **Modified Message**: "U Love Icecream" → Hash → 93218

Even a small change in input causes a **completely different hash**, which is a sign of a good hash function.





Output Sizes of Popular Hash Functions

Hash Function	Output Size
MD5	128 bits (insecure)

Hash Function	Output Size
SHA-1	160 bits (insecure)
SHA-256	256 bits
SHA-3-512	512 bits

Applications of Hash Functions

- **Data Integrity**: Check whether data has been tampered with (e.g., file checksums).
- **Digital Signatures**: Message is hashed before signing.
- **Password Hashing**: Store only hashed passwords for authentication.
- **Blockchain**: Used to link blocks securely.
- **MAC/HMAC**: Message Authentication Codes built from hash functions.

Collision:

A **collision** occurs when:

Hash(msg1) == Hash(msg2) AND msg1 ≠ msg2

This is rare with good hash functions but can be exploited if the function is weak (e.g., MD5, SHA-1).

Important Notes:

- **Hash ≠ Encryption**: No decryption possible, as there's no key.
- **Used with Signatures**: Hashes are often signed with private keys to ensure both **integrity** and **authentication**.

Q. Differentiate encryption vs hashing.

Aspect	Encryption	Hashing
Purpose	To **convert data into unreadable form** to protect privacy	To **generate a unique fixed-length fingerprint** of data
Output	**Ciphertext** – reversible with a key	**Hash digest** – not reversible
Reversibility	**Yes** – decrypt with key	**No** – one-way function

Aspect	Encryption	Hashing
Key Usage	Requires a key (symmetric or asymmetric)	No key needed
Fixed Output Size	Output size depends on input size (unless block cipher)	Always fixed output size (e.g., SHA-256 = 256 bits)
Security Objective	**Confidentiality**	**Integrity**
Used In	File encryption, secure communication (e.g., VPN, HTTPS)	Password storage, digital signatures, checksums
Example Algorithms	AES, DES, RSA	MD5, SHA-1, SHA-256, SHA-3

Q. How do hash functions prevent tampering?

Hash functions are used to **verify the integrity of data**. They generate a **unique fixed-length digest** (hash value) from an input message.
Even a **1-bit change in the input** will produce a **completely different hash**, due to the **avalanche effect**.

How Hashing Detects Tampering

Step-by-step process:

1. **Sender computes** the hash of the original message:

   $$h1=H(Message)$$

2. The message (and possibly the hash) is sent to the receiver.
3. **Receiver computes** their own hash from the received message:

   $$h2=H(Received\ Message)$$

4. Receiver **compares h_1 and h_2**:
   - If $h_1 == h_2$ → message is **unchanged**
   - If $h_1 \neq h_2$ → message has been **tampered with**

Why It Works

Hash Property	Role in Tamper Detection
Deterministic	Same input → same hash
Avalanche Effect	Tiny input change → big output change
Collision Resistance	Hard to create two inputs with the same hash

Even a small alteration (e.g., changing “Hello” to “hello”) causes a **completely different hash**. So tampering is immediately detected.

Q. Types of Authentications

Message Authentication: Two Levels

1. Lower-Level Process

- **Goal**: Generate an **authenticator** to verify the integrity and authenticity of a message.
- The process:
    1. Takes input (like a message and possibly a key).
    2. Produces an **authenticator** (a value used to verify the message).

2. Higher-Level Process

- **Goal**: Use the lower-level primitive (authenticator) to verify the message at the receiver's end.
- Example: Receiver checks the received message using the authenticator to confirm it hasn't been tampered with.

Techniques for Message Authentication

Method	Description
1. Message Encryption	Ciphertext of the plaintext serves as the authenticator. Only someone with the correct decryption key can verify the message.
2. Message Authentication Code (MAC)	A **fixed-length value** generated using a secret key and a public function. It ensures that the message was not altered.
3. Hash Functions	Public functions generate a **fixed-length digest** of the message. Commonly used with digital signatures for integrity.

Types of User Authentication

1. Knowledge-Based Authentication

- Also known as **"Something You Know"**
- Examples:
    - Passwords
    - PINs
- Most common and traditional form of authentication.

2. Possession-Based Authentication

- Also known as **"Something You Have"**
- Examples:
    - Entry/access cards
    - FASTag (RFID)
    - OTP sent to a phone

3. Inherence-Based Authentication

- Also known as **"Something You Are"**
- Examples:
    - Biometric scans (fingerprint, face, retina)
    - Voice or behavioral biometrics

Password-Based Authentication (Digital Example)

1. **User** enters:
    - Username
    - Password
2. The **client application** retrieves the **Distinguished Name (DN)** — a unique identifier string for the user (e.g., in LDAP systems).
3. Client sends:
4. DN + password

to the **server**.

5. Server checks:
    - Whether the DN exists
    - Whether the password matches

6. If both match → **User authenticated**, access is granted.

Q. Explain MAC and how it ensures message integrity.

A **MAC (Message Authentication Code)** is a **short piece of data** generated from a **message and a secret key**, used to:

- **Verify message integrity** (that the message hasn't been tampered with), and
- **Authenticate the sender** (that it came from someone who knows the secret key).

MAC=F(Message,Secret Key)

Purpose:

A **MAC** is used to ensure:

- **Message authenticity**: Confirms the message came from someone who knows the key.
- **Data integrity**: Confirms the message hasn't been altered during transmission.

How MAC Works (Symmetric Key-based)

MAC is a **symmetric key cryptographic technique** — both sender and receiver share the **same secret key K**.

Sender Side:

1. Computes:

   MAC = MAC_Algorithm(K, message)

2. Sends:

   [message + MAC] → to Receiver

Receiver Side:

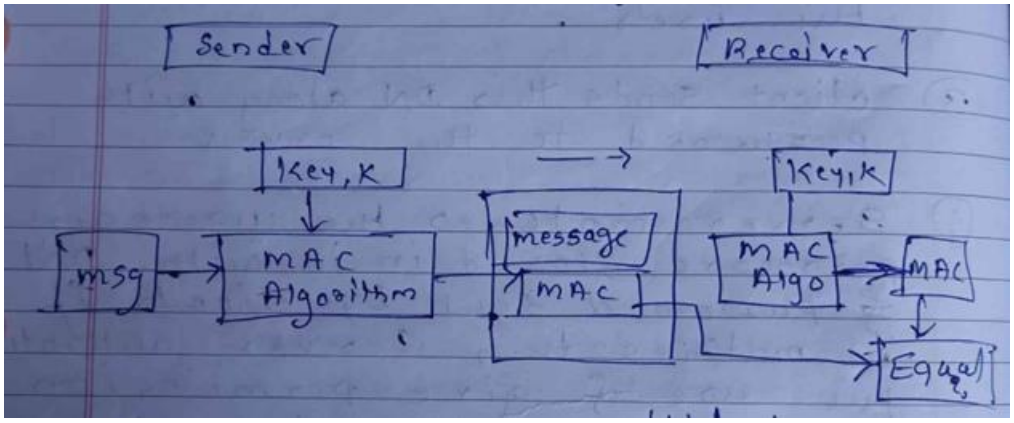
1. Receives the **message and MAC**
2. Uses the **same key K** to compute:

   MAC' = MAC_Algorithm(K, received_message)

3. Compares:
   - If MAC == MAC' → Message is **authentic**
   - If not → Message is **rejected or flagged**

Common MAC Algorithms:

Algorithm	Description
HMAC	Based on hash functions (e.g., HMAC-SHA256)
CMAC	Based on block ciphers (e.g., AES)
UMAC/VMAC	Universal hashing-based MACs (very fast)



How MAC Ensures Integrity

Feature	Explanation
Tamper Detection	If even 1 bit of the message changes, the MAC won't match.
Key Binding	Only someone with the **secret key** can produce a valid MAC.
Recalculation Check	Receiver recalculates MAC to ensure message wasn't altered in transit.

Limitations of MAC:

- **No Non-Repudiation**:
  - Since **both sender and receiver share the same key**, **either could have generated** the MAC.
  - If a dispute arises (e.g., "I didn't send that!"), MAC **cannot prove** the identity of the true sender.
- **Requires Pre-shared Secret Key**:
  - Both parties must securely agree on a key **before** communication begins.

Q. Explain HMAC

HMAC is a **combination of a cryptographic hash function and a secret key**. It provides:

- **Message authentication**
- **Data integrity**
- Strong resistance to common attacks (e.g., collision attacks)

It is **stronger than plain hashing** and **more secure than traditional MACs** in many cases.

How HMAC Works

Formula:

HMAC(K, m) = H[(K ⊕ opad) || H[(K ⊕ ipad) || m]]

Where:

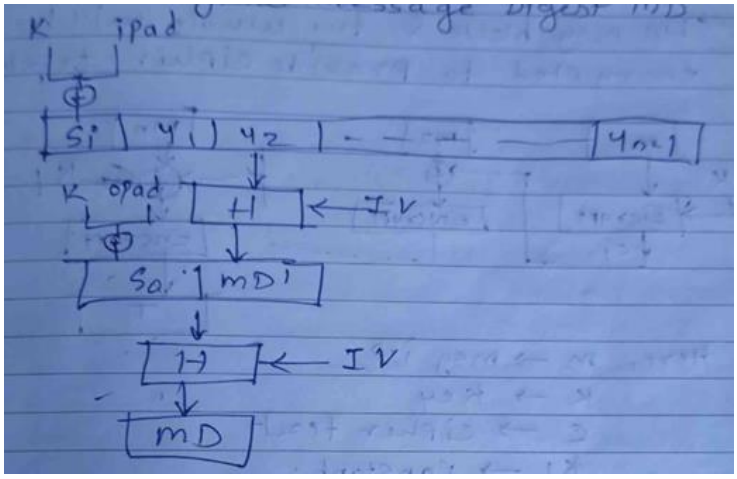
- H = cryptographic hash function (e.g., SHA-256)
- K = secret key
- m = message
- ⊕ = XOR
- || = concatenation
- ipad = inner padding (0x36 repeated)
- opad = outer padding (0x5C repeated)

Step-by-Step Process:

1. **Key Preparation**:
    - If the key K is longer than the block size → hash it
    - If it's shorter → pad it with zeros
2. **Inner Hash**:

   inner = H((K ⊕ ipad) || message)
3. **Outer Hash**:

   outer = H((K ⊕ opad) || inner)
4. **Final Output** = outer hash = HMAC



Why HMAC is Secure:

Feature	Benefit
Uses secret key	Ensures only those with key can generate valid HMAC
Two layers of hashing	Prevents many hash-specific attacks
Resistant to collisions	Even if the hash function is slightly weak, HMAC is still strong

---

Applications:

- Used in **HTTPS**, **TLS**, **IPSec**
- **JWT (JSON Web Token)** for API security
- **Email verification**
- **File integrity checking**
- **Blockchain**

Advantages of HMAC:

1. **High performance**
2. **Compact** compared to digital signatures
3. **Easier key handling** than raw MACs
4. **Stronger against collision attacks** than regular hashing

---

Disadvantages / Limitations:

1. **Shared Key Problem**:
    - Sender and receiver **both know the same key**, so **non-repudiation is not possible**
    - If the key is leaked, **attackers can forge valid HMACs**

Q. Explain CMAC.

CMAC is a secure **message authentication algorithm** built using **block ciphers** like AES. It improves upon the older **CBC-MAC** by preventing its known weaknesses when messages vary in length.

How CMAC Works:

CMAC processes a message in blocks using the **CBC (Cipher Block Chaining)** mode of encryption.

Process Overview:

Input: Message M (divided into blocks m1, m2, ..., mn)

Secret key K

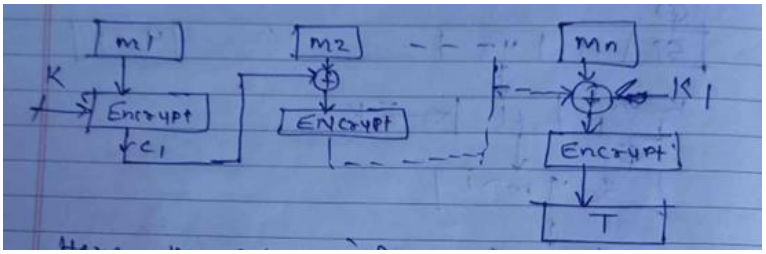
Constants K1 and K2 (derived from K)

Step 1: Divide the message into blocks (mi)

Step 2: XOR each block with the previous ciphertext (CBC-style)

Step 3: Encrypt each result using AES with key K

Step 4: Final encrypted block is the **MAC tag (T)**



Key Components:

- m: Input message (split into blocks)
- K: Secret key
- K1, K2: Subkeys derived from K using a block cipher

- T: Final block of encrypted data = **authentication tag**

CMAC Advantages:

Feature	Benefit
Strong authentication	Based on proven block ciphers (AES)
Fixed-length tag	Output tag is a fixed size (e.g., 128-bit)
No hash function needed	Unlike HMAC, it uses only a cipher
Resistant to CBC-MAC flaws	Secure even with messages of varying lengths

---

Limitations:

- Like other symmetric MACs:
  - **No non-repudiation** (both sender and receiver share the same key)
  - Requires secure key management

Q. Differentiate between HMAC and CMAC.

Aspect	HMAC	CMAC
Full Name	Hash-based Message Authentication Code	Cipher-based Message Authentication Code
Based On	**Hash functions** (e.g., SHA-256, SHA-1)	**Block ciphers** (e.g., AES, 3DES)
Algorithm Type	Uses cryptographic hash functions	Uses symmetric block cipher in CBC mode
Key Type	Symmetric (same key for both sides)	Symmetric
Padding	Uses inner (ipad) and outer (opad) padding	Uses constants (K1, K2) for final block
Output Size	Depends on hash function (e.g., 256 bits for SHA-256)	Same as block size (e.g., 128 bits for AES)
Speed & Efficiency	Fast, especially on platforms optimized for hashing	Fast, especially when AES hardware is available

Aspect	HMAC	CMAC
Security Strength	Strong (secure if hash function is secure)	Strong (secure if block cipher is secure)
Key Size	Varies (depends on hash function, often 128–512 bits)	Same as cipher key (e.g., 128/192/256 bits)
Use Cases	JWT, TLS, HTTPS, API security, file integrity	Wireless comms, IPsec, embedded systems
Standardization	Defined in **RFC 2104**	Defined in **NIST SP 800-38B**
Hardware Support	Less common (though SHA hardware exists)	Widely supported (especially AES-CMAC)
Collision Resistance	Higher if using SHA-2 or SHA-3	Inherits security from cipher (e.g., AES)
Flexibility	Works with many hash functions	Tied to specific ciphers (mostly AES)

Q. Explain MD5 hashing algorithm.

What is MD5?

- A **one-way cryptographic hash function**.
- Takes input of **any length** and returns a **fixed 128-bit** (16-byte) output — called a **message digest**.

Key Properties of MD5:

Property	Description
Input	Message of any length
Output (Digest)	Fixed-length: **128 bits** (32 hex characters)
One-way	Cannot retrieve original message from hash
Deterministic	Same input → same output always
Fast & lightweight	Used widely in earlier systems

Limitations of MD5:

Issue	Explanation
Collision-prone	Two different inputs can produce same hash
Vulnerability to attacks	Not secure for digital signatures or certificates
Short hash length	128 bits is relatively weak by modern standards

MD5 is considered *broken* and should NOT be used for secure applications (e.g., SSL, digital signatures).

Q. Explain SHA hashing algorithm.

Purpose:

SHA-1 is a **one-way cryptographic hash function** designed to:

- Generate a fixed-length **160-bit hash** value from an input of **any length < 2^{64} bits**
- Ensure **message integrity**, authentication, and tamper detection

---

Basic Features:

Feature	Description
Input Length	< 2^{64} bits
Output Length	160 bits (20 bytes, 40 hex chars)
Block Size	512 bits
Internal Word Size	32 bits
Rounds	80 rounds per 512-bit block
Structure	Follows **Merkle–Damgård construction**
Collision-Resistant	**Not anymore** (broken since 2017)

SHA-1 Working Steps:

1. Pre-processing

Before hashing, the input message undergoes formatting:

1. **Padding**:
    - Append a single 1 bit

- Append 0s until the message length ≡ 448 mod 512
- Append 64-bit representation of original message length

2. **Divide into 512-bit blocks**

---

2. Initialize MD Buffer (Hash Values)

Five 32-bit **initial hash values**:

A = 0x67452301

B = 0xEFCDAB89

C = 0x98BADCFE

D = 0x10325476

E = 0xC3D2E1F0

---

3. Processing Each 512-bit Block

Each block goes through 80 rounds:

Step A: Break into 16 words:

- Divide 512-bit block into 16 × 32-bit words: W[0]...W[15]

Step B: Extend to 80 words:

- For t = 16 to 79:

  W[t] = (W[t-3] ⊕ W[t-8] ⊕ W[t-14] ⊕ W[t-16]) <<< 1

Step C: 80 Operations (Rounds)

Each round updates A, B, C, D, E:

For t = 0 to 79:

TEMP = (A <<< 5) + f(t, B, C, D) + E + W[t] + K[t]

E = D

D = C

C = B <<< 30

B = A

A = TEMP

- f(t, B, C, D) is a round-dependent nonlinear function
- K[t] are constant values based on round group

---

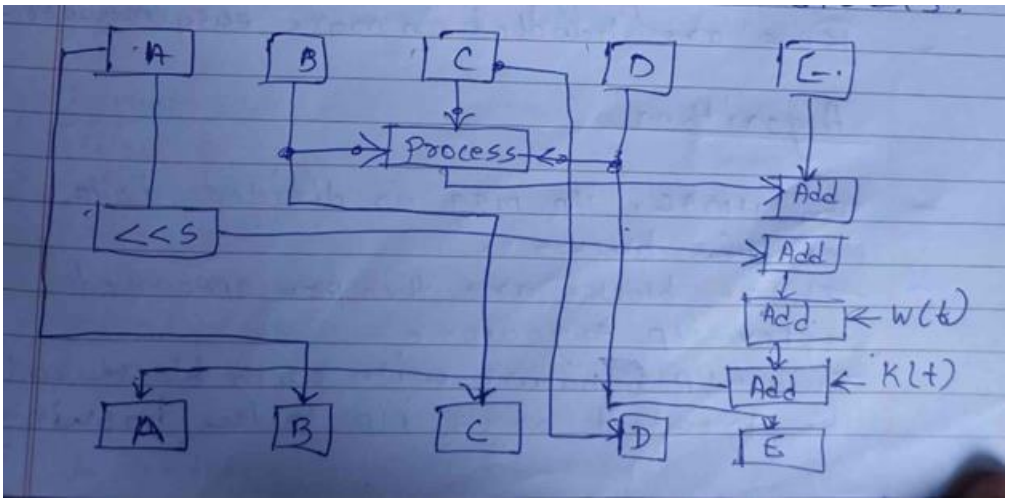
4. Final Output

After all blocks are processed, concatenate the final values of A, B, C, D, and E:

SHA-1 Digest = A || B || C || D || E → 160-bit hash

SHA-1 Security Status:

- **SHA-1 is deprecated** due to successful **collision attacks**
- Google + CWI Amsterdam broke SHA-1 in **2017** with the **SHAttered attack**
- Modern alternatives: **SHA-256, SHA-384, SHA-512, SHA-3**



Summary Flowchart (Simplified)

Message → Padding → Divide into 512-bit blocks

↓

Initialize A, B, C, D, E

↓

For each 512-bit block:

- Expand to 80 words
- Do 80 rounds using A-E

↓

Update A-E for next block

↓

Final A || B || C || D || E = 160-bit Digest

Q. Compare MD5 vs SHA.

Feature	MD5	SHA (SHA-1 / SHA-2 / SHA-3)
Full Form	Message Digest Algorithm 5	Secure Hash Algorithm
Developed By	Ronald Rivest (1991)	NIST (SHA-1: 1995, SHA-2: 2001, SHA-3: 2015)
Output Size	128 bits (32 hex digits)	SHA-1: 160 bits, SHA-2: 224/256/384/512 bits
Security	Broken (collision attacks)	SHA-1 Broken, SHA-2 & SHA-3 Secure
Collision Resistance	Not collision-resistant	SHA-1 Week, SHA-2/3 Strong
Speed	Very fast	Slightly slower (especially SHA-2/3)
Use Cases (Today)	File checksums (non-security only)	Digital signatures, SSL/TLS, blockchain (SHA-2)
Status	Deprecated for security applications	SHA-1 deprecated; SHA-2/3 recommended
Hash Algorithm Type	Merkle–Damgård (uses MD4 internally)	Merkle–Damgård (SHA-1/2), Sponge (SHA-3)

Q. What are digital certificates? Explain X.509 format.

A **digital certificate** is an electronic document used to **prove ownership of a public key**. It helps verify that a public key **truly belongs to the claimed entity** (e.g., a person, website, or organization).

Digital certificates are used in **public key infrastructure (PKI)** systems and are issued by trusted entities called **Certificate Authorities (CAs)**.

Purpose of Digital Certificates

- **Authenticate identity** (e.g., HTTPS website is really who it says it is)

- **Bind a public key** to a verified identity
- **Enable secure communication** (e.g., in SSL/TLS)
- **Prevent impersonation and MITM attacks**

X.509 Certificate Format

X.509 is the most widely used standard for digital certificates, especially in **SSL/TLS (HTTPS)**, **email security**, and **VPNs**.

Components of a Digital Certificate

A digital certificate contains:

Field	Description
Version	Indicates the version of the X.509 standard (usually v3).
Serial Number	A unique identifier assigned by the Certificate Authority (CA) to the certificate.
Signature Algorithm	Specifies the algorithm (e.g., RSA, ECDSA) used by the CA to sign the certificate.
Issuer	The CA that issued the certificate (contains its name and details).
Serial Number	Unique identifier assigned by CA
Not Before	The start date when the certificate is valid.
Not After	The expiration date when the certificate becomes invalid.
Subject	The entity to whom the certificate is issued (e.g., website or individual).
Public Key	The public key associated with the subject.
Issuer ID	Unique identifier for the CA. (optional)
Extensions	Additional information such as Key Usage, SAN (Subject Alternative Name), and other constraints.
Signature	The cryptographic signature from the CA to validate the certificate's authenticity.

How It Works (Simplified)

1. **CA verifies identity** of the subject.

2. CA **issues a certificate**, digitally signs it using its **private key**.
3. Anyone can verify the certificate's authenticity using the **CA's public key**.
4. The public key inside the certificate can now be **trusted**.

Real-World Example

When you visit https://www.example.com, your browser checks the site's **X.509 certificate** to ensure:

- It's issued by a **trusted CA**,
- It hasn't **expired or been revoked**,
- The public key matches the **domain name**.

Q. Differentiate between digital certificate and digital signature.

Feature	Digital Certificate	Digital Signature
Purpose	Verifies **ownership of a public key**	Verifies **authenticity and integrity of a message**
What it contains	Public key, identity info (subject), CA signature	Hash of message **encrypted with sender's private key**
Issued by	A **Certificate Authority (CA)**	Created by the **message sender**
Used for	**Authentication** of a person/domain's public key	**Authenticating messages or documents**
Based on	X.509 standard in Public Key Infrastructure (PKI)	Hashing + asymmetric encryption
Verifies what?	That the public key belongs to the subject	That the message came from the sender and wasn't altered
Reusability	Reused for many transactions	Unique to each message/document
Example Use	HTTPS certificates (e.g., www.google.com)	Signed emails, software packages, blockchain data

Q. How does a Certificate Authority (CA) verify and issue digital certificates?

A **Certificate Authority (CA)** is a **trusted third party** in a Public Key Infrastructure (PKI) that:

- **Verifies the identity** of entities (e.g., websites, individuals, organizations)
- **Issues digital certificates**, binding a **public key** to the verified identity
- Signs the certificate using its **own private key** to ensure trust

---

Steps in Certificate Issuance by a CA

Step	Action
1. CSR	Entity sends public key + identity info to CA
2. Verify	CA validates identity (domain, org, etc.)
3. Sign	CA signs the public key with its private key
4. Issue	CA issues an X.509 certificate to the entity

- The CA ensures **public keys are trusted** and belong to **verified identities**, forming the **foundation of secure internet communication** (e.g., SSL/TLS).

Q. Why are digital certificates needed?

Need	Explanation
Public Key Authentication	Verifies that a public key actually belongs to its claimed owner
Prevents Impersonation	Stops attackers from presenting fake keys
Secure Communication (HTTPS)	Used in SSL/TLS to protect data on websites
Trust Establishment	Rely on trusted Certificate Authorities to vouch for identity
Scalable Key Management	Digital certificates enable public keys to be distributed securely at scale
Enables Digital Signatures	Certificates verify the identity of signers in documents or software

Q. What is PKI, and how does it support trust?

PKI is a **framework** of technologies, policies, and roles used to:

- Manage **digital certificates**
- Distribute and verify **public keys**
- Enable **trusted, secure communication** over networks

It forms the **backbone of trust** in public key cryptography systems such as:

- **HTTPS (SSL/TLS)**
- **Email encryption**
- **VPNs**
- **Digital signatures**

Components of PKI

Component	Role
Certificate Authority (CA)	Trusted entity that issues and signs digital certificates
Registration Authority (RA)	Verifies user identity before certificate is issued
Public & Private Keys	Used for encryption, decryption, signing, and verification
Digital Certificates (X.509)	Bind public keys to verified identities
Certificate Revocation List (CRL) / OCSP	Used to check if a certificate is revoked
PKI Repository	Stores public certificates and CRLs

How PKI Supports Trust

PKI establishes **digital trust** through the following process:

Step 1: Identity Verification

- The **RA** or **CA** verifies the user's identity (e.g., domain ownership, personal ID)

Step 2: Certificate Issuance

- The **CA issues a digital certificate**, binding the public key to the verified identity
- It **digitally signs** the certificate using its **own private key**

Step 3: Certificate Validation

- Other parties (e.g., browsers) **trust the CA's signature**
- If a user presents a certificate signed by a **trusted CA**, they are **automatically trusted**

Step 4: Secure Communication

- Trusted certificates enable:
    - **Encryption** (e.g., using the public key)
    - **Authentication** (proving the certificate owner's identity)
    - **Integrity** (ensuring data isn't tampered with)

Importance of PKI

Benefit	Explanation
Authentication	Verifies the identity of users, websites, or systems using certificates
Confidentiality	Enables **encryption** of data using public keys
Integrity	Ensures data hasn't been altered using **digital signatures**
Non-repudiation	Users can't deny sending signed data (their private key proves it)
Trust	CAs act as trusted authorities in global systems (e.g., HTTPS)

Q.