Digital Certificates & TLS — Deep Dive

From CSR to Certificate Issuance and Client Verification

1) Key Definitions

Digital Certificate (X.509): An electronic document that binds a subject (domain/organization) to a public key. Issued and signed by a Certificate Authority (CA).

Certificate Authority (CA): A trusted entity that validates identities and signs certificates. Root CAs are trusted via OS/browser stores; intermediates are chained to roots.

Hashing: A one-way function (e.g., SHA-256) that maps data to a fixed-length digest. Used for integrity; tiny input changes produce different digests.

Digital Signature: A cryptographic value computed over a hash of data using a private key (RSA/ECDSA/EdDSA). Verifiable by anyone with the corresponding public key.

PEM vs DER: PEM is Base64-encoded text with header/footer lines; DER is the raw binary ASN.1 encoding. Internally, signatures and hashes are over the DER bytes.

2) CSR — Exact Structure & What's Signed

A CSR is an ASN.1 structure with these parts:

• CertificationRequestInfo (plain text)

- subject: DN fields (CN, O, OU, L, ST, C).
- subjectPublicKeyInfo: the applicant's public key (algorithm + key).
- attributes: e.g., extensionRequest (SANs, keyUsage, etc.).

• signatureAlgorithm (plain text)

- Algorithm identifier for the CSR's signature (e.g., sha256WithRSAEncryption).

• signature (plain text bits)

- Digital signature over the DER-encoded CertificationRequestInfo using the applicant's private key.

> **Signed bytes:** exactly the DER of CertificationRequestInfo. The CA recomputes the hash over these exact bytes to verify the CSR signature.

3) Certificate (X.509) — Exact Body (TBSCertificate)

A certificate is a SEQUENCE of three parts. Only the first part (TBSCertificate) is signed. The three parts:

• tbsCertificate (plain text → the part that is signed)

- version, serialNumber.
- signature: algorithm identifier (should match the outer signatureAlgorithm).
- issuer: the CA's distinguished name.
- validity: notBefore, notAfter.
- subject: the end-entity's distinguished name (domain owner).
- subjectPublicKeyInfo: the end-entity public key (algorithm + key).

- extensions: e.g., subjectAltName (DNS names), basicConstraints, keyUsage, extendedKeyUsage, authorityKeyIdentifier, subjectKeyIdentifier, CRL distribution points, AIA (OCSP/Issuers).

• **signatureAlgorithm (plain text)**

- The algorithm the CA used to sign TBSCertificate (e.g., sha256WithRSAEncryption, ecdsa-with-SHA256, RSASSA-PSS).

• **signatureValue (plain text bits)**

- The CA’s digital signature over the DER-encoded TBSCertificate.

Plain text here means 'not encrypted'. It is ASN.1/DER-encoded data visible to anyone. Integrity is provided by the CA’s signature over TBSCertificate.

4) How the CA Creates the Signature (Over TBSCertificate)

Steps the CA performs:

- Choose signature scheme & hash (policy-driven): commonly sha256WithRSAEncryption (OID 1.2.840.113549.1.1.11), ecdsa-with-SHA256 (1.2.840.10045.4.3.2), or RSASSA-PSS.
- Serialize TBSCertificate to DER bytes (exact byte string).
- Compute digest: H = Hash(DER(TBSCertificate)).
- Apply private-key signing to H to obtain signatureValue.

RSA PKCS#1 v1.5 (most common historically)

1) Build ASN.1 DigestInfo = SEQUENCE { AlgorithmIdentifier(hashOID), OCTET STRING(digest) }. 2) EMSA-PKCS1-v1_5 pad DigestInfo to key length with 0x00 0x01 FF..FF 0x00. 3) signature = (padded_block)^d mod n.

RSA-PSS (modern, parameterized)

Uses MGF1 and a salt. Encode H with EMSA-PSS-ENCODE (hash, saltLen, MGF1). Then RSASP1: signature = EM^d mod n.

ECDSA (elliptic curve)

Compute digest H (possibly truncated). Choose random k, compute R = (k·G). Let r = R.x mod n; s = k^{-1}(H + r·d) mod n. Signature is the pair (r, s).

Ed25519 (EdDSA)

Deterministic: derive nonce from private key and message using SHA-512; compute signature (R, S). No separate hash selection; algorithm defines it.

5) How the Client Verifies the Certificate

- Build path: server cert → intermediate(s) → trusted root from OS/browser store. Check basicConstraints (CA=true for issuers), keyUsage/extendedKeyUsage, nameConstraints, pathLen.
- Extract TBSCertificate and compute H' = Hash(DER(TBSCertificate)) using the algorithm in signatureAlgorithm.
- Use issuer’s public key (from the parent certificate in the chain) to verify signatureValue over TBSCertificate.
- For RSA v1.5: compute m = signature^e mod n, parse padded DigestInfo and compare embedded digest with H'.
- For RSA-PSS: verify EMSA-PSS with the given parameters (salt length, MGF1).

- For ECDSA/EdDSA: verify the (r,s)/(R,S) against H' and the issuer public key Q.
- Hostname verification: ensure requested DNS name matches subjectAltName (preferred) or CN.
- Temporal validity: current time ∈ [notBefore, notAfter].
- Revocation: check CRL/OCSP; optionally OCSP stapling provided by the server.
- If all checks pass for each link in the chain, the end-entity certificate is trusted.

Important: The client does NOT obtain the CA’s public key from the server’s certificate. It comes from the trusted root store (directly for a root, or via the parent certificate for an intermediate).

6) Plaintext vs Signed — Quick Reference

- **CSR signed bytes:** DER(CertificationRequestInfo).
- **Certificate signed bytes:** DER(TBSCertificate).
- **Fields outside signature:** signatureAlgorithm and signatureValue (both plaintext).
- **Everything is readable:** 'Plaintext' means not encrypted; confidentiality is not the goal here—integrity and authenticity are.

7) Worked Example (RSA v1.5 with SHA-256)

1. CA policy chooses sha256WithRSAEncryption.
2. CA encodes TBSCertificate to DER → bytes M.
3. Computes digest H = SHA-256(M).
4. Builds DigestInfo with OID(sha256) and H; pads with EMSA-PKCS1-v1_5 to modulus size.
5. Signs: S = EM^d_CA mod n_CA.
6. Client later verifies: compute H' = SHA-256(M). Compute EM' = S^e_CA mod n_CA, parse DigestInfo, confirm OID=sha256 and digest=H'. If equal → signature valid.

Tip: To inspect structures locally

CSR (PEM → text): `openssl req -in server.csr -noout -text`
Certificate (PEM → text): `openssl x509 -in server.crt -noout -text`
Chain building: ensure server sends intermediates so clients can reach a trusted root.