PKI

Motivation and PKI components:
the key distribution problem

Public Key Infrastructure (PKI)

Definition

- A system of policies, roles, software, hardware, and procedures
- Designed to create, manage, distribute, use, store, and revoke digital certificates
- Provides a trust framework for associating public keys with identities

Key components

- Certification Authority (CA) – issues and manages certificates
- Registration Authority (RA) – verifies identities before certificate issuance
- Certificate Repository – stores and distributes certificates and revocation data

The math is not the issue

- RSA, ECC, EdDSA are strong
- Attacks succeed not by breaking math, but:
  - Substituting keys
  - Misleading users
  - Exploiting trust gaps

Assumption of authenticity

- Public-key crypto assumes Alice already knows Bob’s key
- But in practice
  - Keys must be fetched from somewhere
  - That channel may be insecure

Trust models: Web of Trust

Web of Trust (WoT), created for PGP

- Users sign others' keys
  - *If I trust Alice, and Alice trusts Bob ⟹ I may trust Bob*
- Pros: decentralized, no single failure
- Cons: complex, subjective, doesn't scale

TOFU (SSH)

- **Trust On First Use (TOFU)**
- First connection: accept presented key
- Later: warn if key changes
- Pros: simple, works in closed systems
- Cons: **first-contact MITM fatal**, warnings ignored

PKI (X.509)

- Centralized hierarchy with **Certificate Authorities (CAs)**
- **Root CA = trust anchor**; intermediates issue on its behalf
  - **Trust anchor** = a pre-installed, known-good public key (e.g. a Root CA), from which all certificate validation begins
- Pros: global scale, automated validation (browsers/OS)
- Cons: single points of failure; CA compromise can affect millions

Comparing models

- Web of Trust → works in small groups, not globally
- TOFU → simple, limited
- PKI → scalable, centralized

No perfect model

PKI architecture

- Key components of a Public Key Infrastructure
- How certificates are issued and signed
- How identities are verified
- Where certificates are stored and validated
- Goal: establishing secure and scalable digital trust

Overview

- PKI relies on three pillars
  - Certification Authorities (CAs) — issue certificates
  - Registration Authorities (RAs) — validate identities
  - Repositories — publish and revoke certificates
- Together: bind identities to public keys, maintain trust

What is a CA?

- **Certification Authority (CA)** = trusted third party in PKI
- Issues **digital certificates**
  - Electronic documents that bind a public key to an identity
  - Signed by a trusted authority
- Provides assurance: identity to public key
- Core assumption: the CA itself is trusted

Root CA

CA form a hierarchy

- Root CA is the top of hierarchy
- Self-signed certificate
- Distributed in OS, browsers, devices
- Serves as trust anchor (see PKI definition)
- Its signature validates an entire chain
- Typically used only to sign intermediates
- Root private key kept offline
    - Minimize risk of compromise

Intermediate CA

- Delegated authority from Root CA
- Issues certificates for end-entities (servers, users, devices)
- Practical roles:
  - Scalability: distribute workload across many CAs
  - Policy separation: different intermediates for SSL/TLS, code signing, client auth
  - Risk containment: compromise of one intermediate doesn’t destroy trust in the root

CA hierarchy

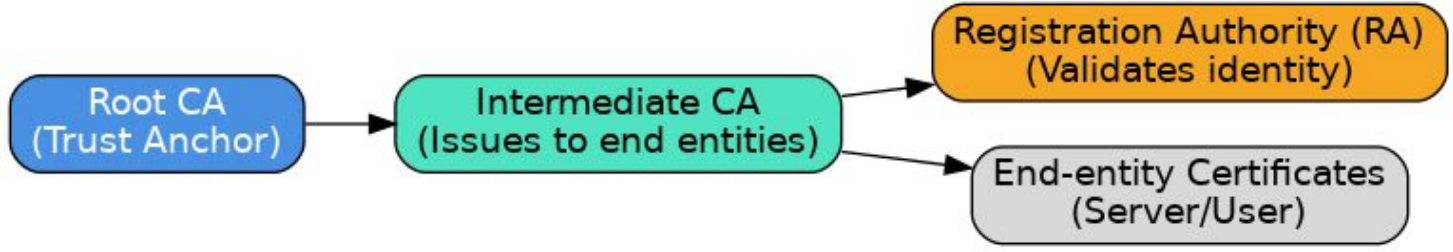
- Root CA → Intermediate CA → End-entity cert
- Each level signs the next one
- Client verifies chain up to Root CA

Why multiple levels?

- **Security**
  - Root key stored offline in HSMs
  - Reduced exposure
- **Flexibility**
  - Different intermediates for different purposes
- **Resilience**
  - One intermediate can be revoked without destroying the root

CA responsibilities

- Validate identities (directly or via RA)
- Issue certificates according to defined policies
- Maintain certificate lifecycle
  - Handle renewals, expirations, and revocations
- Distribute status information
  - Publish CRLs
  - Respond to OCSP requests



What is an RA?

Registration Authority (RA)

- Entity delegated by CA to handle identity vetting
- Ensures only legitimate applicants receive certificates

Role

- Authenticate applicants
- Verify documents, credentials, domains
- Collect registration data
  - Name, organization, domain ownership
- Forward validated requests to CA for issuance

Certificate repository

- Trusted location for storing CA certificates and revocation information
  - Publishes CA and intermediate certificates
  - Publishes revocation data (CRLs, OCSP endpoints)
  - Enables discovery of trust anchors and validation material
- Provided and maintained by the appropriate CA
- Not used to store end-user certificates (privacy + scalability reasons)
- Purpose: ensure that relying parties can retrieve the information required to validate certificate chains and check revocation status

Certificate Revocation List (CRL)

- Digitally signed list of revoked certificates issued by a CA
  - Published periodically
  - Clients download and check status

Limitations of CRLs

- Scalability issues
  - Lists can become very large
  - Not updated in real time → propagation delays
- Performance overhead
  - Downloading and storing CRLs is costly for clients

Online Certificate Status Protocol (OCSP)

- Real-time validation protocol
- Client queries CA/OCSP responder
  - “Is this cert valid?” → Yes/No
- More efficient than downloading full CRLs
- OCSP stapling

Why rooted trust is important

- PKI depends on pre-installed trusted roots
- Every certificate chain terminates at a root
- Without trusted roots
  - No secure certificate validation
  - Encryption and authentication lose reliability

Modern trust hierarchies

- Level 1: Root CA
  - Self-signed, ultimate trust anchor
  - Kept offline, used only to sign intermediates
- Level 2: Intermediate CA (subordinate to root)
  - Issues end-entity or further subordinate certificates
  - Enables scalability and policy separation
- Level 3+: Subordinate / additional intermediates
  - May issue end-entity certs (servers, users, devices)
  - Can delegate further (rare in practice)
- In practice
  - Typical chain: Root → Intermediate → End-entity
  - Longer chains exist in large or government PKIs

How many root CAs?

- Per PKI hierarchy
  - Exactly one Root CA
  - All intermediates and end-entities trace back to it
- OS / browsers
  - Trust stores include 100–200+ root CAs
  - Each belongs to a different CA operator (e.g., DigiCert, GlobalSign, Let’s Encrypt)
- Private / enterprise PKI
  - Often 1–3 root CAs for redundancy or policy separation
  - Still, each hierarchy has only one trust anchor
- Key point
  - A single PKI → 1 root
  - A device/browser trusts many roots simultaneously

What is a trust anchor?

- Pre-installed, known-good public key/certificate
- Distributed with OS and browsers
- Starting point for validation
- All chains checked against the anchor

Properties of trust anchors

- Self-signed (no higher CA signs them)
- Pre-installed in OS and browser trust stores
- Embedded directly in systems by vendors
- Long validity periods
  - Typically 10–25 years
  - Rarely replaced
  - Renewal requires vendor updates

Security of trust anchors

- Stored in highly secure environments (data centers, vaults)
- Operated mostly offline to reduce exposure
- Keys protected inside Hardware Security Modules (HSMs)
  - Tamper-resistant devices for private key storage
- If compromised
  - Attacker can issue valid-looking certificates
  - All trust chains under that root are broken

Case of DigiNotar (2011)

- Dutch CA – compromised
- Impact
  - Large-scale MITM attacks (notably in Iran)
  - Browsers revoked DigiNotar roots → CA collapsed
  - Lesson: Root CA compromise undermines global digital trust
- Over 500 certificates fraudulently generated
- Targeted major global domains
  - Google (incl. *.google.com, Gmail)
  - Yahoo
  - Facebook
  - Microsoft (Hotmail/Live)
  - Twitter
- Also issued for government and intelligence agencies
  - CIA, MI6, Mossad

What is a Trust Store?

- Database of trusted root certificates
- Maintained by
  - Operating systems
  - Browsers
- Defines which roots are trusted

Management & governance

- Vendors decide admission/removal
- Audits and compliance rules
- Enterprises can
  - Add internal CAs for VPNs
  - Remove untrusted roots
- Risk: misconfiguration can block services

Risks of Trust Store manipulation

- Malware may install rogue roots
- Enables silent HTTPS interception
- User awareness is low
  - Attackers gain undetectable MITM capability

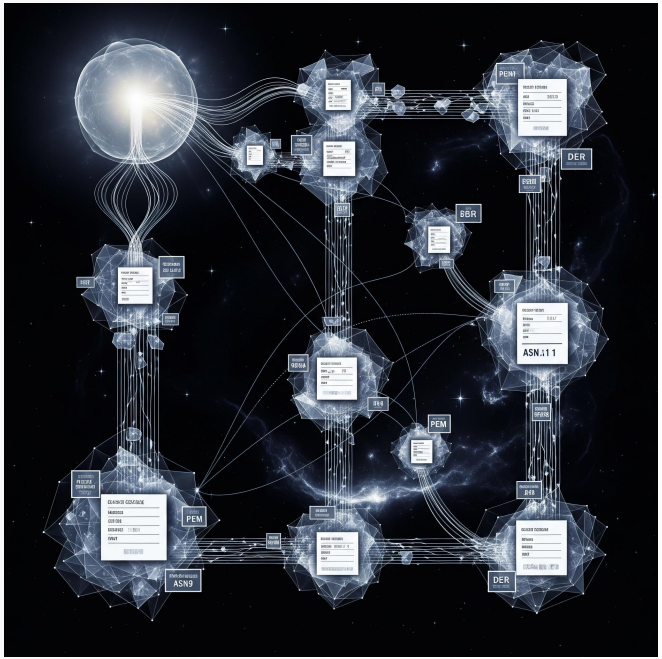
Certificates are signed

- **Self-Signed Roots**
  - Signed with their own private key
  - Distributed via OS and browser trust stores
  - Foundation of the PKI trust model

- **Issued Certificates**
  - Signed by a higher CA (root or intermediate)
  - Inherit trust from the chain up to the root
  - Examples
    - Server TLS certificate
    - S/MIME certificate
    - Code signing certificate

PKI

Structure of X.509 certificates:
encoding formats



Why encoding matters

- Certificates = digital identity containers
- Need strict encoding rules for
  - Compatibility across platforms
  - Unambiguous interpretation
  - Reliable signature verification

X.509 and encoding

- X.509 = certificate standard
- But: standard defines fields, not bits on the wire
- Solution: ASN.1 + encoding rules (DER/PEM)
- Encodings are essential to make certificates usable

Introduction to ASN.1

- Abstract Syntax Notation One (ASN.1)
  - Formal language for defining data structures
  - Defined in the ITU-T X.680 series standards
  - First standardized version of the OSI Abstract Syntax Notation (1980s)
- Applications
  - Telecom signaling protocols and networking standards
  - Public Key Infrastructure (e.g., X.509 certificates)
  - Data exchange in cryptographic protocols
- Key Benefits
  - Platform-neutral
  - Language-independent
  - Flexible extension mechanism (supports versioning and backward compatibility)

ASN.1 in X.509

- Every field in a certificate is defined in ASN.1
  - Official definition: a certificate is a SEQUENCE of three fields

```
Certificate ::= SEQUENCE {
  tbsCertificate       TBSCertificate,
  signatureAlgorithm   AlgorithmIdentifier,
  signatureValue       BIT STRING
}
```

fields types

- Each field is defined using an ASN.1 type

Three-layer model

- X.509 (conceptual): defines the certificate as an object
- ASN.1 (syntax): defines abstract structure
- DER/PEM (encoding): defines concrete representation
- Hierarchical relationship ensures clarity and consistency

DER and PEM

DER (Distinguished Encoding Rules)

- Binary encoding of ASN.1 structures
- Strict, compact, unambiguous
- File extensions: .der, .cer, .crt
- Example (binary bytes): 30 82 03 9F 30 82 02 87 A0 03 ...

PEM (Privacy Enhanced Mail)

- Base64 encoding of DER + header/footer
- Human-readable, easier to copy/paste
- File extensions: .pem, .crt, .cer, .key
- Example (text form)

```
-----BEGIN CERTIFICATE-----
MIIDdzCCAl+gAwIBAgIEb1Cb...
-----END CERTIFICATE-----
```

BER vs DER

- BER (Basic Encoding Rules)
  - Original ASN.1 encoding (ITU-T X.690)
  - Multiple valid encodings for the same structure
  - Flexible, efficient for streaming, but not canonical

- DER (Distinguished Encoding Rules)
  - Canonical subset of BER: only one encoding allowed
  - Required for signatures (bits must match exactly)

DER: core features

- Binary format, compact and efficient
- Strict rules for determinism
- Used for
  - Signature generation & verification
  - Storage in trust stores (OS/browsers)

PEM vs DER

DER
- Binary
- Smaller size
- Used by software

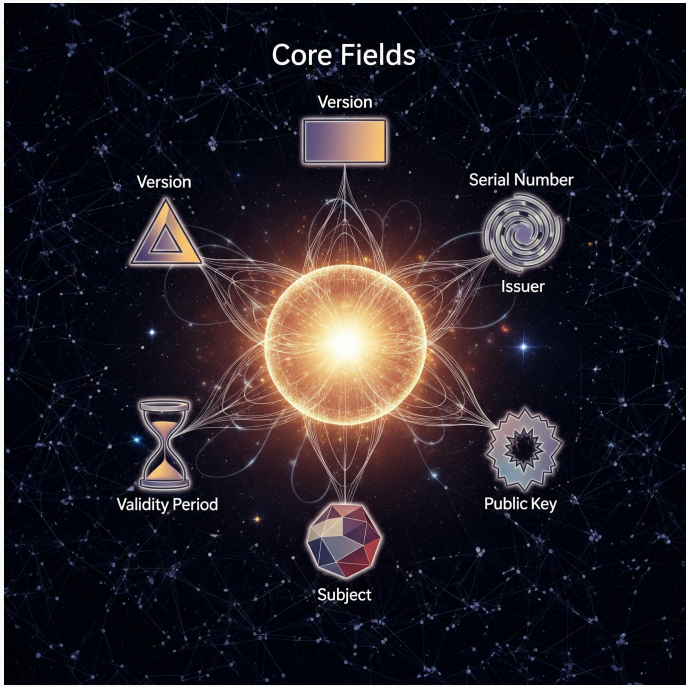
PEM
- ASCII Base64
- Larger but portable (≈133%)
- Used in web servers, email, certificate bundles

Security implications

- DER ensures exact byte representation ⇒ safe for signing
- PEM must always decode back to DER before validation
- Attack surface: malformed ASN.1/DER encodings (historical OpenSSL bugs)

PKI

Structure of X.509 certificates: core fields



Core fields of X.509 certificates

- Defined in TBSCertificate structure (RFC 5280)
- Capture identity, key material, and validity
- Foundation of digital trust

Overview of TBSCertificate

SEQUENCE with multiple fields

- Version (v1, v2, v3)
- Serial Number (unique per issuer)
- Signature (AlgorithmIdentifier)
  - Declares the algorithm intended for signing
  - Must match the outer Certificate.signatureAlgorithm
- Issuer (CA name)
- Validity (Not Before / Not After)
- Subject (entity identified by the certificate)
- Subject Public Key Info
  - Algorithm (e.g., RSA, EC)
  - Public Key value
- [IssuerUniqueID – optional, v2+]
- [SubjectUniqueID – optional, v2+]
- [Extensions – optional, v3+]

Version field

- Indicates X.509 version
  - v1: original (few fields, no extensions)
  - v2: added unique identifiers (rarely used)
  - v3: adds extensions (dominant today)
- Encoded as INTEGER (0=v1, 1=v2, 2=v3)

Serial number

- Unique identifier assigned by issuer CA
- Must be unique per CA → prevents reuse
- Used in
  - Revocation lists (CRLs, later)
  - Certificate validation and debugging
- Typically, large random integer

Signature Algorithm (in TBSCertificate)

- Declares the algorithm to be used for signing/verifying the certificate
  - Examples
    - sha256WithRSAEncryption
    - ecdsaWithSHA384
- Must match the outer field Certificate.signatureAlgorithm
- This field is part of the **signed data** (inside TBSCertificate), while the actual signature value is stored separately in signatureValue

Issuer field

- Distinguished Name (DN) of the entity that signed the certificate (usually a CA)
- Represented as an ASN.1 SEQUENCE of Relative Distinguished Names (each RDN is a set of attributes)
  - Common attributes (not all mandatory):
    - C = Country
    - O = Organization
    - OU = Organizational Unit (optional)
    - CN = Common Name
    - L = Locality, ST = State/Province (optional)
- In a Root CA certificate: issuer = subject → self-signed, acts as trust anchor

About DNs

- **Hierarchical** naming system (derived from X.500)
  - X.500 defines a hierarchical, directory-based naming system for uniquely identifying entities, from which X.509 and LDAP are derived
- Composed of flexible attributes (RDNs), e.g.
  - CN=DigiCert Global Root CA, O=DigiCert, C=US
  - CN=Example Intermediate, O=Example Corp, C=IT
  - CN=Alice, OU=Research, O=Example Corp, C=IT
- During certificate validation, DNs are parsed and matched to build the trust chain

Why hierarchical?

C=US
└── ST=California
    └── L=San Francisco
        └── O=Example Corp
            └── OU=IT Department
                └── CN=Alice Smith

CN=Alice Smith, OU=IT Department, O=Example Corp,
L=San Francisco, ST=California, C=US

Validity field

- Defines time window certificate is valid
- Two fields
  - notBefore (start time)
  - notAfter (end time)
- Enforced strictly: expired → rejected
- Format: GeneralizedTime (YYYYMMDDHHMMSSZ)
  - Z = UTC

Validity in practice

- Typical lifetimes (approximate ranges):
  - Root CA: 15–25 years (sometimes up to 30)
  - Intermediate CA: 5–10 years (commonly)
  - TLS server certificates: maximum 398 days (CA/Browser Forum rule)
    - ≈ 1 year + 1 month
- Shorter validity reduces risk exposure and enforces key rotation
- General principle: shorter validity = better security

Subject field

- Identifies the entity being certified
- Represented as a Distinguished Name (DN)
- Common attributes
  - CN = Common Name (traditionally hostname, deprecated in TLS, replaced by SAN (cert extens., later)
  - O = Organization
  - OU = Organizational Unit
  - C = Country, ST = State/Province, L = Locality (optional)
- In modern TLS, Subject may be empty (SAN-only certificates)
- Browsers validate hostname strictly against SAN, not CN

Subject Public Key Info (SPKI)

- Encapsulates the subject's public key and the parameters of its algorithm
- Structure (OID=Object Identifier)
  - AlgorithmIdentifier (OID + parameters)
  - SubjectPublicKey (BIT STRING)
- Examples
  - RSA → OID + NULL params
  - EC → OID + curve OID
- Used to
  - Validate signatures created with the subject's private key
  - Encrypt data for the subject (RSA only)

Relationship of core fields

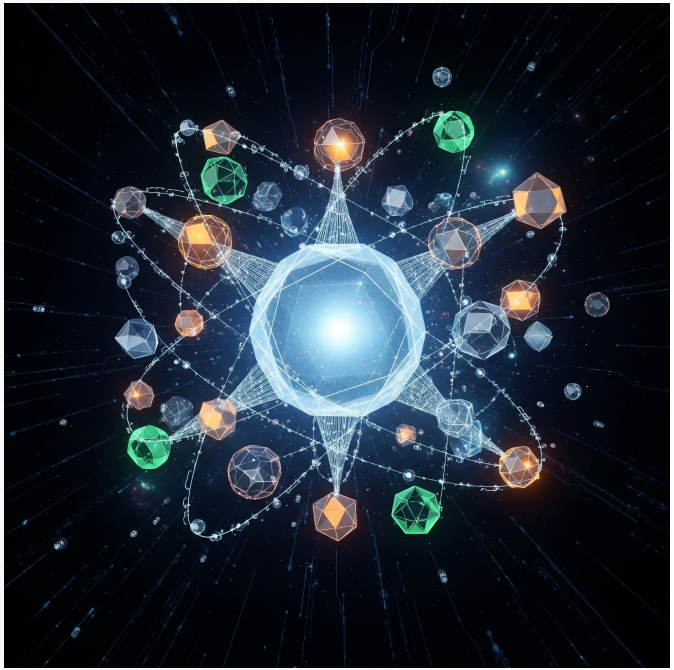
- Issuer signs the TBSCertificate (includes Subject’s identity, public key, validity, serial, extensions)
- Serial number uniquely identifies this certificate within the Issuer’s scope
- Validity defines the trusted time window
- All together: digital certificate = identity + key + timeframe + issuer’s signature

Common pitfalls

- Serial number reuse
  - breaks revocation
- Incorrect validity
  - expired or not yet valid certs
- Wrong subject fields (CN mismatch)
  - hostname errors
- Weak signature algorithm
  - insecure trust

PKI

Structure of X.509 certificates: extensions



X.509 certificate extensions

- Introduced in X.509 v3
- Purpose: extend core certificate information
- Enable
  - Richer identity binding
  - Usage restrictions on keys
  - Additional trust controls

Why extensions?

- Core fields provide: subject, issuer, validity, public key
- But modern PKI requires
  - Multi-domain support (SAN)
  - Usage constraints (Key Usage / EKU)
  - Delegation and path length limits (Basic Constraints)
  - Chain building aids (SKI/AKI)

Extension model (RFC 5280)

- Each extension has three elements
  - extnID = OID uniquely identifying the extension (e.g., KeyUsage, SAN)
  - critical = Boolean flag
    - if TRUE, extension must be understood and enforced, or the cert is rejected
  - extnValue = ASN.1 encoded content of the extension
- Allows extensibility → vendors can define new OIDs

Critical vs non-critical

- Critical extensions
  - Must be understood and enforced → otherwise the certificate is invalid
  - Example: Basic Constraints in CA certificates
- Non-critical extensions
  - Can be ignored if not recognized
  - Example: Certificate Policies, CRL Distribution Points
- Mis-marking = interoperability or security problems
- Every extension must explicitly be marked as critical or non-critical

Subject Key Identifier (SKI)

- Identifies the certificate's public key
- Typically: SHA-1 hash of subjectPublicKey (per RFC 5280 recommendation)
- Benefits
  - Simplifies certificate chain building (used together with AKI in child certs)
  - Enables unambiguous key matching
  - Helps clients select the correct issuer
- Usually marked as non-critical

Authority Key Identifier (AKI)

- Identifies the issuer’s public key
- Main field: keyIdentifier (usually equals the SKI of the issuer’s certificate)
- Optional fields: issuer’s name and certificate serial number
- Used to link an end-entity or intermediate certificate to the correct issuing CA
- Typically marked as non-critical

SKI-AKI relationship

- Validation = AKI (child) ↔ SKI (issuer)
- Helps when
  - Multiple possible issuers exist
  - Trust store has many candidates
  - Avoids ambiguity compared to using Issuer DN only
- SKI and AKI are typically non-critical, but essential for chain building

Key usage

- Restricts operations allowed with the key
- Common flags
  - digitalSignature → TLS handshake (ECDHE signatures), code signing
  - keyEncipherment → RSA key transport (encrypt session key)
  - keyAgreement → DH/ECDH key exchange
  - keyCertSign → sign other certificates (CA only)
  - crlSign → sign CRLs (CA only)
- Security: certificate invalid if key used outside allowed purpose
- Enforced by TLS libraries and operating systems
- RFC 5280 explicitly recommends marking key usage as critical when the certificate’s use must be restricted

Extended Key Usage (EKU)

- Defines specific application purposes (refines Key Usage)
- Examples (identified by OIDs)
  - TLS Web Server Authentication
  - TLS Web Client Authentication
  - Email Protection (S/MIME)
  - Code Signing
  - Time Stamping
- Certificates may contain multiple EKUs
- In public PKI: usually non-critical for compatibility
- In private PKI: can be marked critical to enforce strict application-specific usage

Subject Alternative Name (SAN)

- Expands subject identity beyond CN
- SAN types
  - DNS names (most common for HTTPS websites)
  - IP addresses (used for devices/services without DNS)
  - Email addresses (in S/MIME for secure email)
  - URIs (used for services like OCSP responders)
- Mandatory in TLS (CN deprecated)
- Marked critical in many implementations, though technically it doesn’t have to be

SAN examples

- Web servers (HTTPS)
  - DNS: www.example.com, example.org
- Multi-domain certs: multiple SAN entries
  - Wildcards: *.example.com (broader risk if compromised)
  - IP-based certs: 192.0.2.1
- Other applications
  - Email certificates: email SANs
  - Mail servers (SMTP/IMAP): DNS or IP SANs
  - VPNs / IoT devices: IP SANs

Basic constraints

- Defines whether a certificate can act as a Certification Authority (CA)
- Fields
  - CA=true/false
  - pathLenConstraint → limits depth of CA hierarchy
- Mandatory in CA certificates
- Enforced during chain validation
- Critical in CA certs (to prevent misuse)
- Non-critical in end-entity certs

Common pitfalls

- CA cert issued without basic constraints: treated as end-entity
- End-entity cert erroneously marked CA=true: can issue other certs
- Incorrect pathLenConstraint: broken validation

Attack scenario

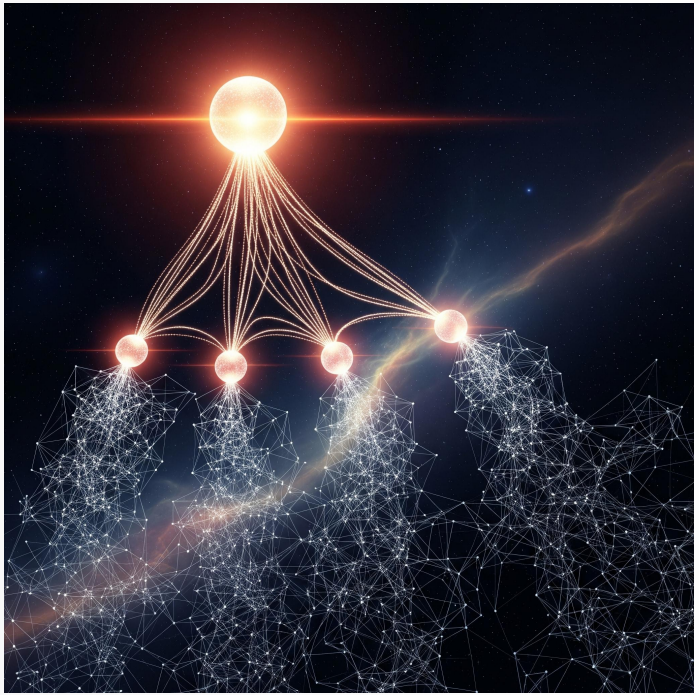
- Real cases: intermediate certs without Basic Constraints
- Exploit: attacker uses it as a rogue CA
- Result: fake but valid-looking certs for any domain, meaning total CA compromise
- Example: 2001 Microsoft sub-CA incident

Other important extensions

- Revocation info
  - CRL Distribution Points (CRL DP): location of CRL
  - OCSP responder URL
- Issuer metadata
  - Authority Information Access (AIA)
  - Issuer CA URL
- Policies
  - Certificate Policies
  - OIDs + CPS reference (Certification Practice Statement)
- Restrictions
  - Name Constraints (limit domains allowed)
  - Subordinate CA scope limitation (enterprise PKI)

PKI

Trust hierarchies and certificate chains: root and intermediate CA



Root CAs as Trust Anchors

- Self-signed certificates
  - Signature created and verified with the same keypair
- Distributed via trust stores
  - Pre-installed in OS, browsers, mobile devices
- Unique position
  - No higher authority, forms the trust anchor
- Validation starting point
  - Certificate chain verification begins with the root

Distribution of root certificates

- Delivered by
  - Operating systems (Windows, macOS, Linux distributions)
  - Browsers (e.g., Firefox maintains its own trust store)
- Trust stores regularly updated via OS/browser updates
- When a root is removed from the store, every certificate chaining to it fails validation

Security of root keys

- Protection measures
  - Root keys kept offline (air-gapped)
  - Stored in Hardware Security Modules (HSMs)
  - Activated only during controlled signing ceremonies
  - Enforced by strict operational and security procedures
- Impact of compromise
  - Breach of a root key undermines the entire trust hierarchy

Intermediate CAs

- Definition
  - Certificates issued by a Root CA or another Intermediate CA
- Characteristics
  - Signed by a higher CA (not self-signed)
  - Validity period shorter than that of the root
  - Operate as delegated CAs, not as trust anchors
- Primary role
  - Issue certificates for end-entities (servers, clients, code signing, etc.)

Purpose of intermediate CAs

- Risk reduction
  - If an intermediate is compromised, only that layer is revoked
- Operational flexibility
  - Different intermediates for TLS, code signing, and document signing
- Scalability
  - Enables large ecosystems while keeping the root offline and secure

Specialized intermediate CAs

- Placement
  - May be issued directly by the Root (Level 2) or by another Intermediate (Level 3)
- Types
  - TLS intermediates
    - For server authentication (HTTPS)
  - Code signing intermediates
    - For signing software applications
  - Document signing intermediates
    - For legal and eID contexts
- Benefit
  - Granular delegation enables tighter control and risk limitation

Certificate types in the PKI hierarchy

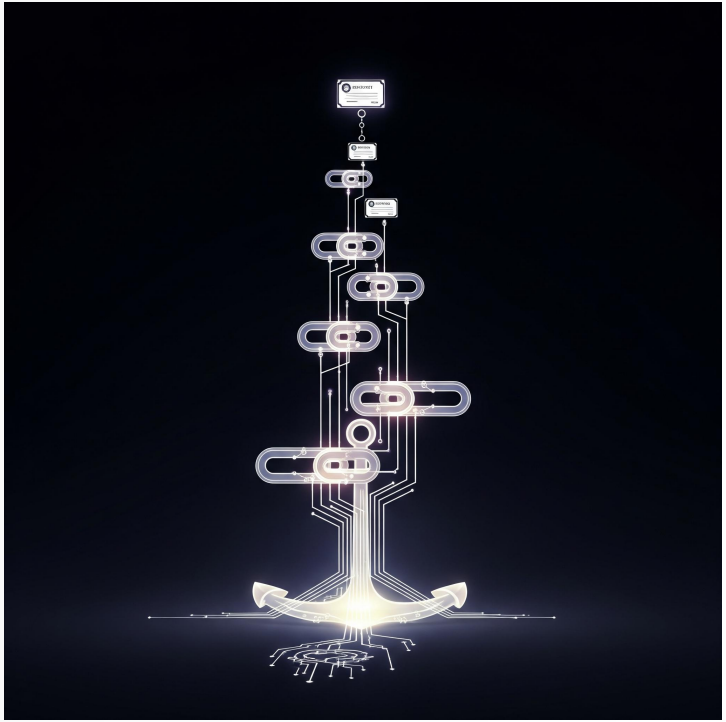
- Root CA certificate
  - Self-signed (trust anchor)
  - Pre-installed in trust stores
- Intermediate CA certificate
  - Signed by a Root (or higher CA)
  - Distributed with end-entity certificates to build trust chains
- End-entity certificate
  - Issued by an Intermediate
  - Used by servers, clients, or applications

Certificate chain validation

- Browser receives
  - End-entity certificate (server)
  - Intermediate certificate(s)
- Builds the chain
  - End-entity
  - Intermediate
  - Root in trust store
- Validates each link
  - End-entity signed by Intermediate
  - Intermediate signed by Root
  - Root trusted in store
- If the chain ends in a trusted root, the certificate is accepted

PKI

Trust hierarchies and certificate chains: chain of trust



Chain of trust: introduction

- Definition: verifiable sequence of certificates from end-entity to trusted root CA
- Purpose: ensure authenticity via digital signature validation
- Components
  - End-entity certificate (server, client, or code signer)
  - Intermediate CAs (delegated authorities extending trust)
  - Root CA (self-signed trust anchor in system/browser store)

End-entity certificates

- Issued to
  - Servers (TLS for secure websites)
  - Users (S/MIME, client authentication)
  - Software publishers (code signing)
- Leaf nodes in the chain, they cannot issue further certificates
- Usage restricted via extensions
  - Key Usage (allowed cryptographic operations)
  - Extended Key Usage (specific applications such as TLS server or email protection)

Intermediate certificates

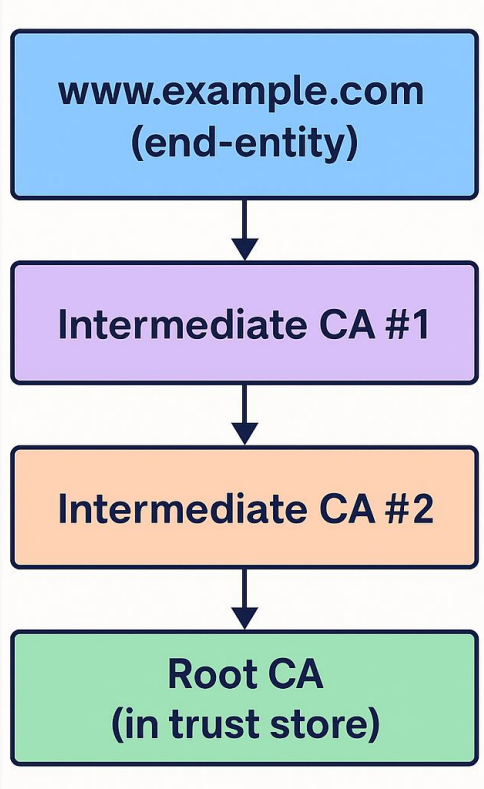
- Delegated by the root CA to improve scalability and reduce risk
- Characteristics
  - Issued by a root or another intermediate CA
  - Marked with CA=true in the Basic Constraints extension
  - May include path length constraints to limit delegation depth
  - Enable building hierarchies while keeping the root offline

Root certificates

- Self-signed and serve as anchors of trust
- Pre-installed in trust stores by operating systems and browsers
- Rarely used to issue end-entity certificates directly (kept offline for security)
- Removal results in immediate distrust of all certificates in the hierarchy

Visualizing the chain

Each link verified
by digital signature

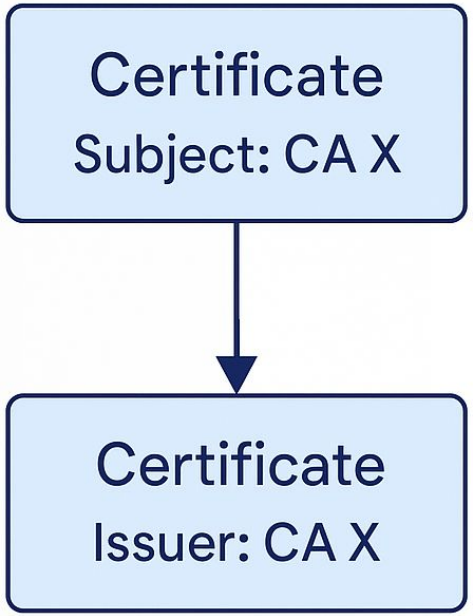


Matching SKI and AKI

- Subject Key Identifier (SKI) uniquely identifies the public key in a certificate
- Authority Key Identifier (AKI) references the public key of the issuer
- Chain construction relies on SKI–AKI matching
- Prevents ambiguity when multiple potential issuers exist in large trust stores

Issuer/subject matching

- Each certificate contains
  - Issuer field: Distinguished Name (DN) of the signer (CA)
  - Subject field: Distinguished Name (DN) of the entity identified
- Validation step
  - The Issuer of a child certificate must match the Subject of its parent certificate
- Combined with SKI/AKI, this ensures structural and cryptographic consistency of the chain



Multiple intermediates

- Certificate chains can include multiple intermediate CAs
- Benefits
  - Segregation of roles (e.g., TLS vs. code signing)
  - Reduces the scope of impact in case of compromise
- Challenge
  - Longer chains increase the number of potential failure points

Chain length and constraints

- Basic Constraints extension
  - CA=true means the certificate can issue other certificates
  - Path length specifies the maximum number of allowed intermediates
- Prevents unrestricted delegation of authority
- Ensures predictable and controlled chain hierarchies

Why not trust intermediates directly?

- Intermediate CAs are not trust anchors
- Their legitimacy derives from being issued by a root certificate
- Directly trusting an intermediate introduces risks
  - Admins bypass vendor trust store policies
  - Weakens revocation mechanisms and governance controls
    - Root vendors can revoke or distrust intermediates, but direct trust bypasses this safeguard

Chain building process (high-level)

- Start with the end-entity certificate presented by the server or client
- Look up the issuer
  - Check provided intermediates (e.g., in TLS handshake)
  - If missing, retrieve issuer via AIA (Authority Information Access)
- Continue building until
  - A trusted root is reached, or
  - Validation fails

Risks in the chain

- Weakest link principle
  - A single vulnerable intermediate compromises the entire chain
- Historical incidents
  - Mis-issued intermediates exploited in MITM attacks
  - Longer chains increase the attack surface

PKI

Certificate lifecycle: revocation

Certificate revocation — definition and purpose

- Revocation = process of invalidating a certificate before its scheduled expiry
- Common reasons
  - Private key compromise
  - Change in subject identity information
  - Certificate issued in error
  - Termination of organization or service
- Revocation status must be published by the CA so clients can detect and reject revoked certificates

Revocation mechanisms

- CRL (Certificate Revocation List)
- OCSP (Online Certificate Status Protocol)
- Enhancements: OCSP stapling, Must-Staple, CRLite
- Each mechanism involves trade-offs in scalability, latency, and reliability

Certificate Revocation Lists (CRLs)

- Signed list of revoked certificates, published by the CA
- Distributed via HTTP, LDAP, or other repositories
- Location included in certificate extension: CRL Distribution Points (CDP)

CRL structure (defined in ASN.1 – X.509)

- Metadata: version, issuer, thisUpdate, nextUpdate
  - nextUpdate = expected time of the next CRL; may be earlier, but not later
- Revoked certificates include
  - Serial number
  - Revocation date (may precede thisUpdate)
  - Revocation reason (optional)
- Entire CRL is encoded in DER and digitally signed by the issuing CA

CRL operation

- Clients download the CRL from the CDP (distribution point)
- Verify the CA's digital signature on the CRL
- Check whether the serial number of the certificate being validated is present in the list
- If the serial number is listed, the certificate is revoked/invalid

CRL drawbacks

- Size: can grow to several MB in large CAs
- Latency: clients may not always fetch the latest CRL
- Scalability: frequent updates create overhead
- Unsuitable for high-volume, real-time services

Delta CRLs

- Contain only changes since the last full CRL
- Reduce size and bandwidth usage
- Clients must support both the base CRL and the delta CRL
- Identified by the deltaCRLIndicator extension, which specifies the number of the base (full) CRL
- Add complexity and are not widely adopted

Online Certificate Status Protocol (OCSP)

- Defined in RFC 6960
- Client sends query to CA's OCSP responder about a specific certificate
- Possible responses:
  - good → certificate is valid
  - revoked → certificate is invalid
  - unknown → CA has no information about the certificate

OCSP structure

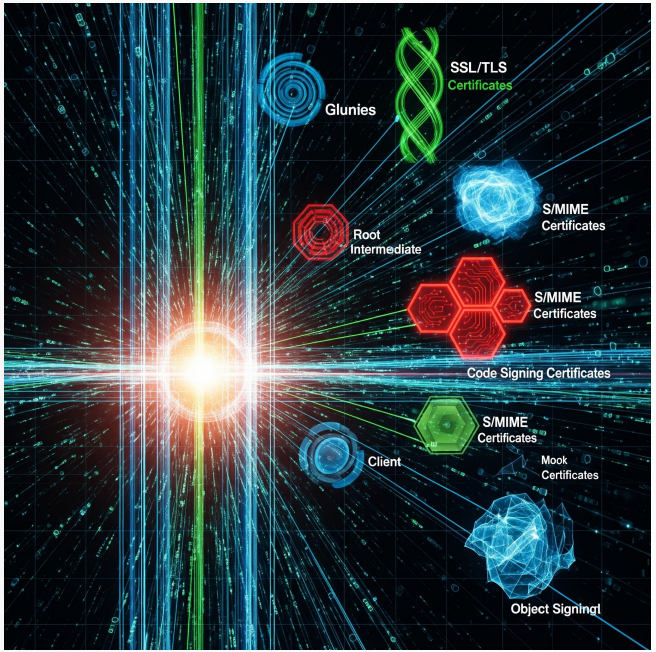
- Request
  - certificate serial number
- Response
  - Cert status (good/revoked/unknown)
  - ThisUpdate, NextUpdate
  - Signed by OCSP responder certificate

OCSP stapling

- TLS server retrieves the OCSP response in advance
- The response is included (“stapled”) in the TLS handshake (CertificateStatus message)
- Server must refresh the stapled response before nextUpdate
- Advantages
  - Eliminates client–CA query latency
  - Preserves user privacy
  - Reduces load on CA responders

PKI

PKI in action: types of certificates



Types of digital certificates

- Certificates differ by purpose and validation level
- Applied to
  - Individuals
  - Servers and organizations
  - Software and devices
- Provide
  - Authentication (prove identity)
  - Confidentiality (encrypt data)
  - Integrity (prevent tampering)

SSL/TLS Certificates

- Secure websites and online services
- Provide
  - Encryption (keeps traffic confidential)
  - Authentication (verifies domain/server identity)
  - Integrity (ensures data is not altered)
- Foundation of HTTPS and modern e-commerce

TLS Certificate Types (by validation)

- **Domain Validation (DV)**
  - Confirms only domain ownership (via email or DNS)
  - Automated, fast, low-cost
  - Lower assurance
- **Organization Validation (OV)**
  - Confirms domain and organization identity
  - Verified against official business records
  - Medium assurance, higher trust than DV
- **Extended Validation (EV)**
  - Strongest vetting: legal, physical, operational checks
  - Historically displayed organization name in browser
  - Higher cost, stricter issuance process

TLS certificate types (by scope)

- **Single-Domain Certificates**
  - Protect exactly one FQDN (e.g., www.example.com)
- **Wildcard Certificates**
  - Secure all subdomains under a domain
  - Example: *.example.com → covers mail.example.com, shop.example.com
- **SAN / Multi-Domain Certificates**
  - Secure multiple domains and subdomains in a single certificate
  - Example: example.com, example.org, mail.example.net

Client certificates

- Used to identify individuals or devices
- Common use cases
  - Employee authentication in enterprises (mutual TLS)
  - VPN access control
  - Secure email (S/MIME client identity)
  - Multi-Factor Authentication (MFA) with certificate + password/token
  - Adoption in zero-trust architectures

Code signing certificates

- Authenticate software publishers
- Ensure integrity of applications (detect tampering or malware injection)
- Commonly required for
  - Mobile apps (iOS, Android)
  - Desktop applications (Windows, macOS)
  - Drivers (Windows kernel mode → EV code signing)
- Key benefits
  - Prevent distribution of malicious or altered software
  - Increase user trust and confidence

Email certificates (S/MIME)

- Enable secure email communication
- Provide
  - Encryption of email contents (confidentiality)
  - Digital signatures for authenticity and integrity
- Common in corporate and government environments
- Alternative: PGP-based certificates

Device & IoT certificates

- Secure devices and machine-to-machine communication
- Commonly used in
  - IoT ecosystems (smart home, wearables)
  - Industrial systems (ICS/SCADA)
  - Smart cards, hardware tokens, TPMs
  - Automotive (connected cars, V2X communication)
- Designed to support large-scale deployments (millions of devices)

Specialized Certificates

- Qualified Certificates (eIDAS – EU regulation)
  - Legally recognized electronic signatures
  - Equivalent to handwritten signatures
  - Used in finance, healthcare, public sector
- Attribute Certificates
  - Do not bind identity to a key
  - Bind roles or permissions (e.g., Doctor, Manager)
  - Useful in access control models
- Short-Lived Certificates
  - Valid for hours or days
  - Reduce reliance on CRLs/OCSP
  - Common in automated systems (e.g., ACME, Kubernetes)

Self-signed certificates

- Signed with their own private key, not by a trusted CA
- Pros
  - Useful for development and testing
  - Suitable for internal systems without external CA
- Cons
  - Not trusted by browsers/OS by default
  - Vulnerable to MITM attacks if not carefully distributed
- Sometimes used within a **private PKI**

Other types

- Code signing
- Email (S/MIME)
- Device & IoT
- Specialized
- Self-signed