The X.509 standard and PKI

Antonio Lioy

< lioy @ polito.it >

Politecnico di Torino

Dipartimento di Automatica e Informatica

Public-key certificate (PKC)

- **definition:**
  - a data structure to securely bind a public key to some attributes
- **"securely bind"**
  - typically with the signature by an authority
  - … other techniques possible (e.g. blockchain, direct trust and personal signature)
- **"some attributes"**
  - those employed in the transaction being protected by the PKC
  - sometimes difficult to decide a-priori, great variability
- **important to achieve non repudiation of a digital signature**
- **PKC is the public complement of the corresponding personal private key**

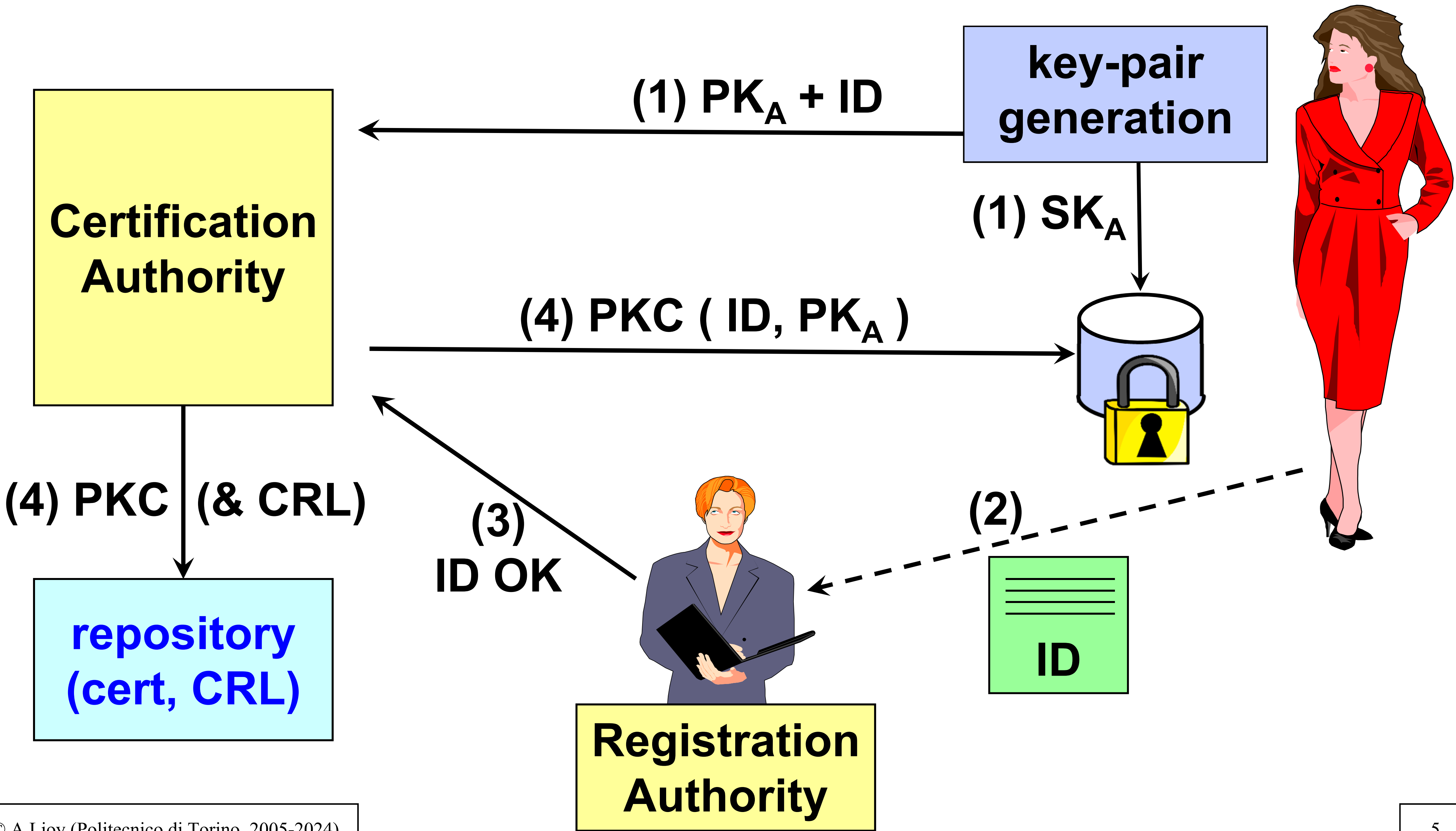
Asymmetric key-pair generation (SK + PK)

- **key generation requires complex algorithms and often RNG**
- **… and after generation, we need to protect the private key**
  - when stored
  - when used
- **generation/use in a software application (e.g. a web browser)**
  - trust in the computing platform (malware? weak implementation?)
- **dedicated hardware (e.g. smart-card)**
  - difficult update of algorithms and mechanisms
  - difficult or impossible vulnerability patching
- **keys may be generated in software and injected into a device**
  - useful for key recovery … but IFF restricted to encryption keys

Certification architecture

- **Certification Authority (CA)**
  - generates / revoke PKC
  - publishes PKC and information about their status
- **Registration Authority (RA)**
  - verifies claimed identity and attributes
  - authorizes PKC issuing / revocation
- **Validation Authority (VA)**
  - provides services to verify the validity status of a PKC (e.g. CRL download and/or OCSP responder)
- **Revocation Authority (unofficial term, role can be assigned to RA or CA)**
  - delegated to timely revoke PKC (more urgent than issuing)

Certificate generation



Certificate generation

- **other architectures are possible**
- **RA generates key-pair, obtains PKC, and distributes them on a secure device (e.g. smart-card)**
  - typical for large companies, where employees are known
- **user first visits the RA and gets a code to be used for authenticating her request to the CA**
  - code = MAC(K, ID)   (where K is a shared secret key CA-RA)

X.509 certificates

- **standard ITU-T X.509:**
  - v1 (1988)
  - v2 (1993) = minor
  - v3 (1996) = v2 + extensions + attribute certificate v1
  - v3 (2001) = v3 + attribute certificates v2
- **is part of the standard X.500 for directory services (white pages)**
- **is a solution to the problem of identifying the owner of a cryptographic key**
- **definition in ASN.1 (Abstract Syntax Notation 1)**

PKC scope

- the certificate contains information that allows to uniquely associate a cryptographic key to an entity
- the binding is guaranteed by a Trusted Third Party (TTP) usually called certification authority (CA), which digitally signs each certificate
- liability may be limited to specific applications or purposes (as specified in the CA's certification policy)

CP and CPS

- **RFC-3647 "Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework"**
- **Certificate Policy (CP)**
  - a named set of rules that indicates the applicability of a PKC to a particular community and/or class of application, with common security requirements
- **Certification Practice Statement (CPS)**
  - a statement of the practices employed by a CA in issuing PKC
- **a CP specifies minimum requirements and can be followed by many CAs (e.g. a government CP to be implemented by all certification providers)**
- **a CPS contains implementation details and is specific of a single CA**

X.500 directory service

- **first application of the X.509v1 certificates**
- **three main problems encountered:**
  - lack of guarantees on the quality of the CA (policy?)
  - lack of an X.500 infrastructure (access to certificates?)
  - difficulty to establish the certification path among two arbitrary users (relationship among the CAs?)

Remedies for X.509v1

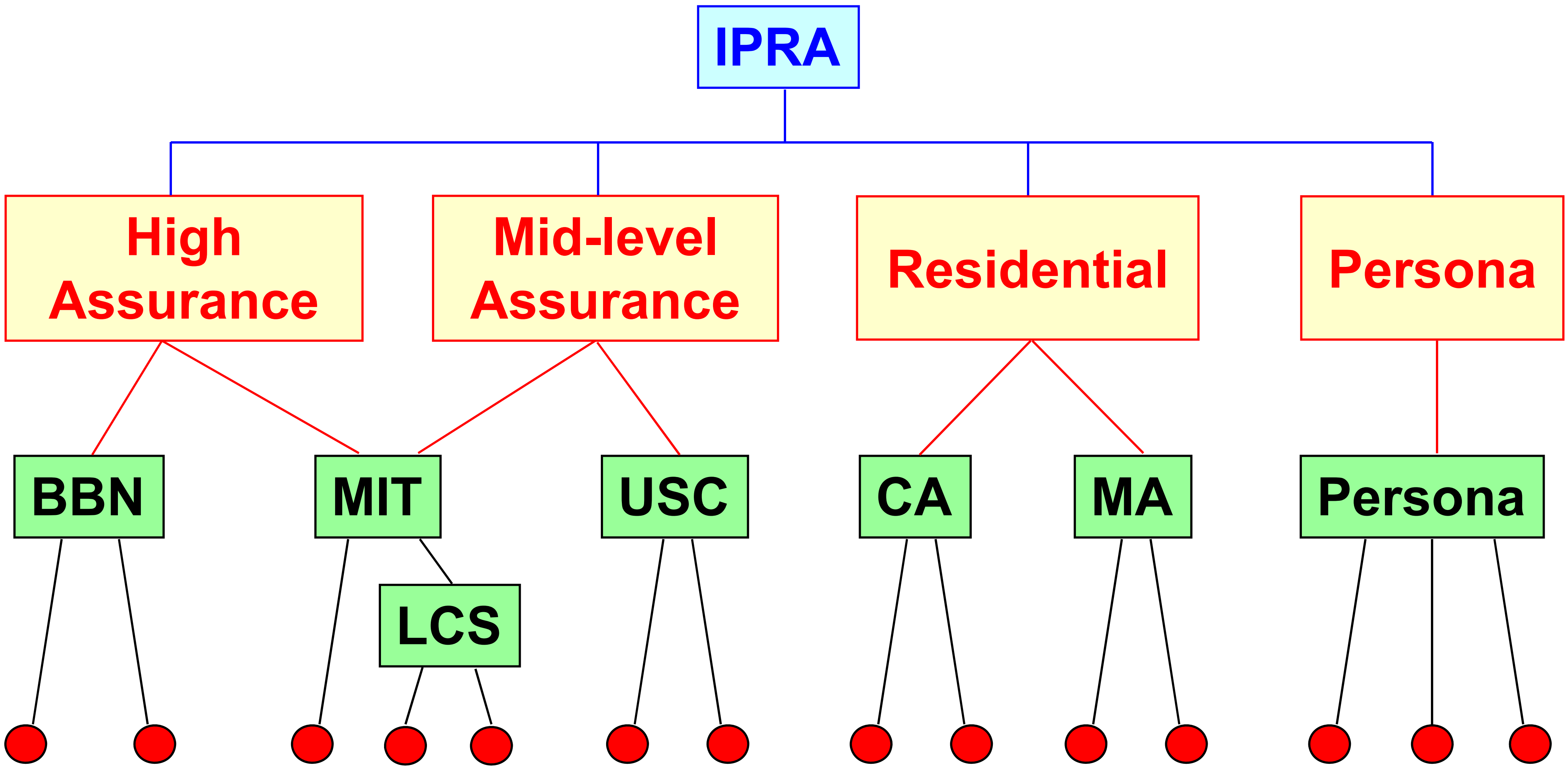
- **force the semantics in the application or in any case in the context external to the certificate**
  - example: RFC-1422 (PEM)

- **make the certificate more flexible and expressive:**
  - X.509 version 3

RFC-1422

- **hierarchical certification infrastructure**
  - rooted at IPRA (Internet Policy Registration Authority)
- **definition of special certification authorities:**
  - PCA (Policy Certification Authority)
  - establish the policies used to issue the certificates
- **name subordination of the CAs (subset of names), for example:**
  - [CA n.1] C=IT
  - [CA n.2] C=IT, O=Politecnico di Torino

Internet PEM hierarchy (RFC-1422)



Problems with RFC-1422

- the hierarchical infrastructure limits the flexibility
- the name subordination introduces undesired limits to the assignment of X.500 names
- the use of the PCA concept is not flexible in commercial applications, where the participation of an operator to take a decision is impractical

X.509 version 3

- **standard completed in June 1996**
- **joint work of ISO/ITU and IETF to define certificates useful for Internet applications**
- **groups together in a unique document the modifications required to extend the definition of certificate and CRL**
- **two types of extensions:**
  - public, that is defined by the standard and consequently made public to anybody
  - private, unique for a certain user community
- **certificate profile**
  - set of extensions for a specific purpose
  - e.g. RFC-5280 "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile"

Base syntax of a X.509 cert

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

TBSCertificate ::= SEQUENCE {
 version              [0] Version DEFAULT v1,
 serialNumber         CertificateSerialNumber,
 signature            AlgorithmIdentifier,
 issuer               Name,
 validity             Validity,
 subject              Name,
 subjectPublicKeyInfo SubjectPublicKeyInfo,
 issuerUniqueID       [1] IMPLICIT UniqueIdentifier OPTIONAL,
                      -- if present, version must be v2 or v3
 subjectUniqueID      [2] IMPLICIT UniqueIdentifier OPTIONAL,
                      -- if present, version must be v2 or v3
 extensions           [3]  Extensions OPTIONAL
                      -- if present, version must be v3
}
```

Critical extensions

- **an extension can be defined as critical or non-critical:**
  - in the verification process the certificates that contain an unrecognized critical extension MUST be rejected
  - a non-critical extension MAY be ignored if it is unrecognized
- **the different (above) processing is entirely the responsibility of the entity that performs the verification: the Relying Party (RP)**

Public extensions

- **X.509v3 defines four extension classes:**
  - key and policy information
  - certificate subject and certificate issuer attributes
  - certificate path constraints
  - CRL distribution points

Key and policy information

- authority key identifier
- subject key identifier
- key usage
- private key usage period
- certificate policies
- policy mappings

Key and policy information

- **authority key identifier (AKI)**
  - identifies a specific public key used to sign a certificate
  - identification by means of:
    - a key identifier (typically the digest of the PK)
    - the pair issuer-name : serial-number
- **use: the same CA could use two keys (e.g. low and high assurance)**
- **non-critical**
- **in some applications, it is very important to build the certificate chain**

Key and policy information

- **subject key identifier**
  - identifies a specific public key used in an application (for example when the public key is updated)
  - non-critical

Key and policy information

- **key usage (KU)**
  - identifies the application domain for which the public key can be used
  - can be critical or non-critical
  - if it is critical then the certificate can be used only for the scopes for which the corresponding option is defined

Key and policy information

- **key usage – the permitted cryptographic operations that can be defined are:**
  - digitalSignature (CA, user)
  - nonRepudiation (user)
  - keyEncipherment (user)
  - dataEncipherment
  - keyAgreement
    (encipherOnly, decipherOnly)
  - keyCertSign (CA)
  - cRLSign (CA)

Key and policy information

- **private key usage period**
  - defines the usage period of the private key
  - extension always non-critical
  - usage is discouraged

Key and policy information

- **certificate policies**
  - list of the policies followed when the certificate was issued and the purposes for which it can be used
  - indication by means of OID, URI, text message
  - critical or non-critical
  - the use of this extension may support not only authentication but also authorization

Key and policy information

- **policy mappings**
  - indicates the correspondence (mapping) of policies among different certification domains
  - present only in the CA certificates
  - non-critical

Certificate subject and certificate issuer attributes

- subject alternative name
- issuer alternative name
- subject directory attributes

Subject Alternative Name (SAN)

- **allows to use different formalisms to identify the owner of the certificate (e.g. e-mail address, IP address, URL)**
- **always critical if the field subject-name is empty**

Issuer Alternative Name (IAN)

- **allows to use different formalisms to identify the CA that issued a certificate or a CRL (e.g. e-mail address, IP address, URL)**
- **always critical if the field issuer-name is empty**

X.509 alternative names

- **various possibilities:**
  - rfc822Name
  - dNSName
  - iPAddress
  - uniformResourceIdentifier
  - directoryName
  - X400Address
  - ediPartyName
  - registeredID
  - otherName

Certificate subject and certificate issuer attributes

- **subject directory attributes**
  - allows to store certain directory attributes associated to the owner of the certificate
    - for example DoD uses it for “citizenship”
  - it's actual usage heavily depends upon the application (as no standard definitions exist)
  - non-critical

Certificate path constraints

- basic constraints
- name constraints
- policy constraints

Certificate path constraints: BC

- **basic constraints**
  - indicates if the subject of the certificate can act as a CA:
    - BC=true : the subject is a CA
    - BC=false : the subject is an EE (End Entity)
  - furthermore, it is possible to define the maximum depth of the certification tree (only if BC=true)
  - critical or non-critical
  - it is suggested to always mark this extension as critical

Certificate path constraints: NC

- **name constraints**
  - only in CA certificates
  - space of names that can be certified by a CA
    - same format as for Alternative Names
  - at least one specification between:
    - permittedSubtree (i.e. whitelist)
    - excludedSubtree (i.e.blacklist)
  - whitelist processed always first
    - beware! an unspecified format in whitelist (e.g. directoryName) is implicitly permitted
  - critical or non-critical (note: no NC support by Apple)

Certificate path constraints: PC

- **policy constraints**
  - used by a CA to specify the constraints that could require an explicit identification by a policy or that inhibit the policy mapping for the rest of the certification path
  - critical or non-critical

CRL distribution point

- **CRL distribution point (aka CRLDP or CDP)**
  - identifies the distribution point of the CRL to be used in validating a certificate
  - can be:
    - directory entry
    - e-mail or URL
  - critical o non-critical

Private extensions

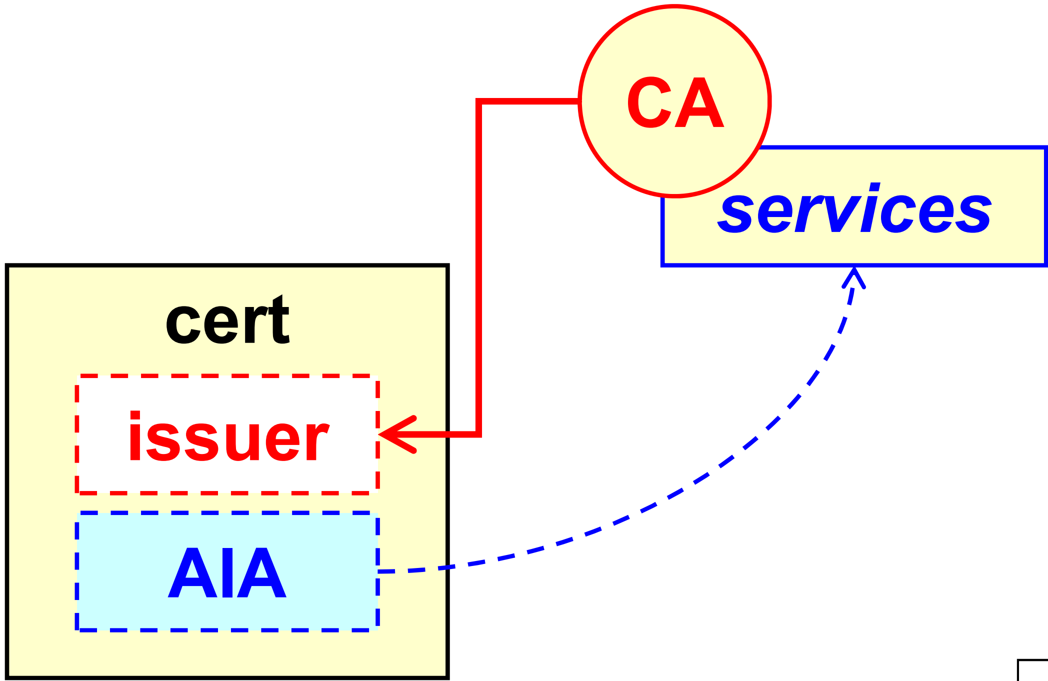
- **it is possible to define private extensions, that is extensions common to a specific user community (i.e. a closed group)**
- **for example IETF-PKIX defined three private extensions for the Internet user community:**
  - subject information access
  - authority information access
  - CA information access

PKIX private extensions

- **subject information access**
  - specifies a method (e.g. http, ldap) to obtain information about the owner of a certificate and a name that indicates the location (address)
  - useful especially when a directory is not used for certificate distribution
  - non-critical

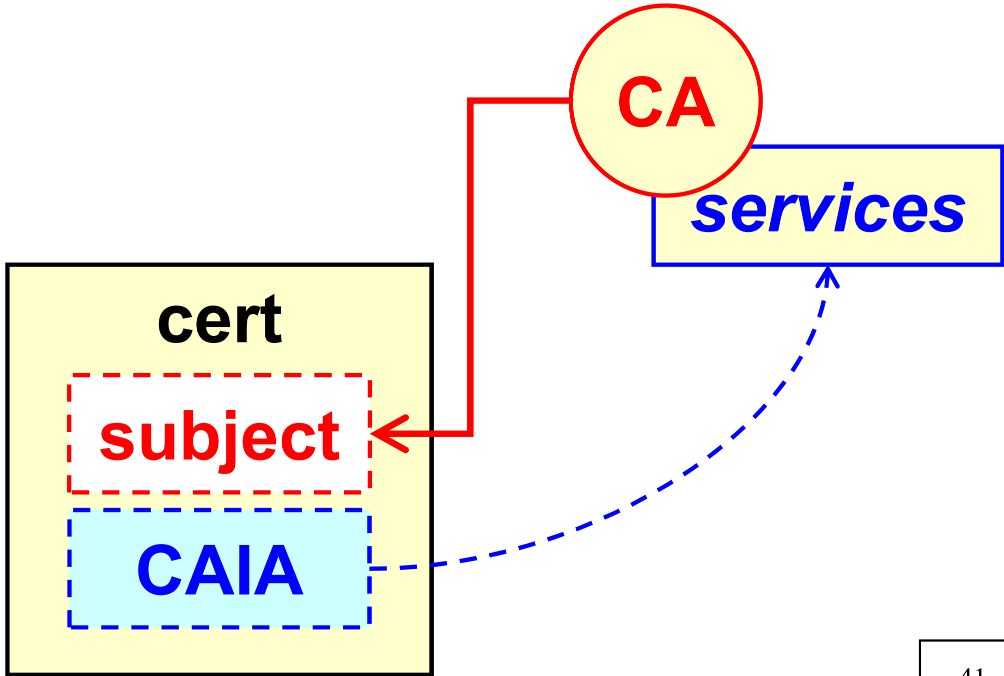
PKIX private extensions: AIA

- **authority information access**
  - indicates how to access information and services of the CA that issued the certificate:
    - certStatus (e.g. URL for OCSP)
    - certRetrieval
    - cAPolicy
    - caCerts
  - critical or non-critical



PKIX private extension: CAIA

- **CA information access**
  - indicates how to access information and services of the CA that owns the certificate:
    - certStatus
    - certRetrieval
    - cAPolicy
    - caCerts
  - valid only in a CA certificate
  - critical or non-critical



RFC-2459

- **profile of X.509v3 suggested by PKIX for use in Internet applications (e.g. IPsec, TLS, S/MIME)**
- **extensions defined by OID with base**
  ID-PKIX ::= 1.3.6.1.5.5.7
  (iso.org.dod.inet.sec.mechanisms.pkix)
- **supported algorithms**
- **implementation suggestions:**
  - e.g. UTCtime
  - mandatory to put seconds
  - mandatory Zulu time
  - if year on two digits = 1950 - 2049

Extended key usage

- **in addition or in substitution of keyUsage**
- **oriented towards application rather than cryptographic operations**
  - may be used simultaneously (priority in case of conflicts?)
- **possible values:**
  - (id-pkix.3.1) serverAuth [DS, KE, KA]
  - (id-pkix.3.2) clientAuth [DS, KA]
  - (id-pkix.3.3) codeSigning [DS]
  - (id-pkix.3.4) emailProtection [DS, NR, KE, KA]
  - (id-pkix.3.8) timeStamping [DS, NR]
  - (id-pkix.3.9) ocspSigning [DS, NR]

Evolution of RFC-2459

- **RFC-2459 was replaced by RFC-3280 + RFC-3279**
- **RFC-3280 defines the Internet profile of PKIs based on certificates X.509v3 and CRL X.509v2**
  - later obsoleted by RFC-5280
- **RFC-3279 documents the algorithms (identifiers, parameters and encodings) used by RFC-3280**
  - includes and makes obsolete RFC-2528 (use of KEA)

RFC-3279

- **algorithms that MUST be supported by the applications that use the RFC-3280 profile**
- **digest:**
  - MD-2, MD-5, SHA-1 (preferred)
- **signing of certificates / CRL:**
  - RSA, DSA, ECDSA (Elliptic Curve DSA)
- **keys of the subject (SubjectPublicKeyInfo)**
  - RSA, DSA, KEA, DH (Diffie-Hellman), ECDSA, ECDH (Elliptic Curve DH)

- **nota: blue algorithms added in RFC-3279 w.r.t. RFC-2459**

Additions to RFC-3279

- **RFC-4055**
  - RSA Probabilistic Signature Scheme (RSASSA-PSS) signature
  - RSA Encryption Scheme - Optimal Asymmetric Encryption Padding (RSAES-OAEP) key transport
  - SHA-2 functions in PKCS#1 signatures
- **RFC-4491**
  - GOST R 34.10-94 / 34.10-2001 / 34.11-94
- **RFC-5480**
  - ECC keys (any, ECDH, ECMQV)
- **RFC-5758**
  - DSA with SHA-224 and SHA-256; ECDSA with SHA-2
- **RFC-8692**
  - RSASSA-PSS and ECDSA with SHAKE128 and SHAKE256

RFC-3280 / -5280 (I)

- **Internet PKI profile based on X.509v3 certificates and X.509v2 CRL**
- **details the path validation algorithm**
  - RFC-2459 had only a high level description
- **specifies the algorithm used for the verification of the status of a certificate by using CRLs**
  - algorithm not present in RFC-2459
- **specifies the algorithm for the use of Delta-CRL**
  - algorithm not present in RFC-2459

RFC-3280 /-5280 (II)

- **adds an ExtendedKeyUsage**
  - (id-pkix.3.9) OCSPSigning [DS, NR]
- **new extensions for certificates:**
  - inhibit any-policy
  - freshest CRL
  - subject information access
- **new extensions for CRL:**
  - freshest CRL

Evolution of RFC-5280

- **RFC-6818 “Updates to the Internet X.509 PKI certificate and CRL profile”**
  - acceptable encoding for explicitText of the user notice policy
  - conversion of internationalized domain name labels to ASCII
  - clarifications for self-signed certificates and trust anchors
- **RFC-8398 “Internationalized email addresses in X.509 certificates”**
- **RFC-8399 “Internationalization updates to RFC-5280”**
  - Internationalized Domain Addresses (IDN) and email addresses

New public PKIX extensions

- **freshest CRL**
  - = Delta CRL Distribution Point
  - same syntax as “CRL Distribution Point”
  - can be inserted either in a certificate or in a CRL
    - but it must not be present in a Delta CRL, so it's always CRL = base + delta (i.e. we don't make delta of delta)
  - non-critical

Certificate revocation

- **a certificate may be revoked before its natural expiration**
  - upon request of the certificate owner (subject)
    - typically due to key compromise or loss
  - upon request of the certificate sponsor (organization)
    - employee going out of the company … or company going out of business
    - dismissed server
  - autonomously by the issuer
    - typically due to issuance error or fraud
- **certificate status MUST be checked by the entity that accepts it to protect a transaction (e.g. commercial order)**
  - this is the relying party (RP)

Mechanisms for checking certificate status

- **do consider the whole chain up to a trusted root**
  - any certificate expired?
  - all certificates valid?
- **if not expired a PKC is valid unless otherwise stated**
- **two possible mechanisms to check validity status:**
  - **CRL (Certificate Revocation List)**
    - list of revoked certificates (i.e. check it by yourself)
    - signed by the issuer or one delegated authority
  - **OCSP (On-line Certificate Status Protocol)**
    - answer about validity of one specific PKC at the current time
    - normally signed by the server providing the answer (trusted?)

X.509 CRL

- **Certificate Revocation List**
- **list of revoked certificates**
  - along with revocation date and reason
- **CRLs are issued periodically and maintained by the certificate issuers**
  - need to re-issue a CRL even if no additional revocation performed, just to ensure its "freshness" and avoid replay attacks with an old CRL
- **CRLs are digitally signed:**
  - by the CA that issued the certificates
  - by a revocation authority delegated by the CA
    - then the CRL becomes an indirect CRL (iCRL)

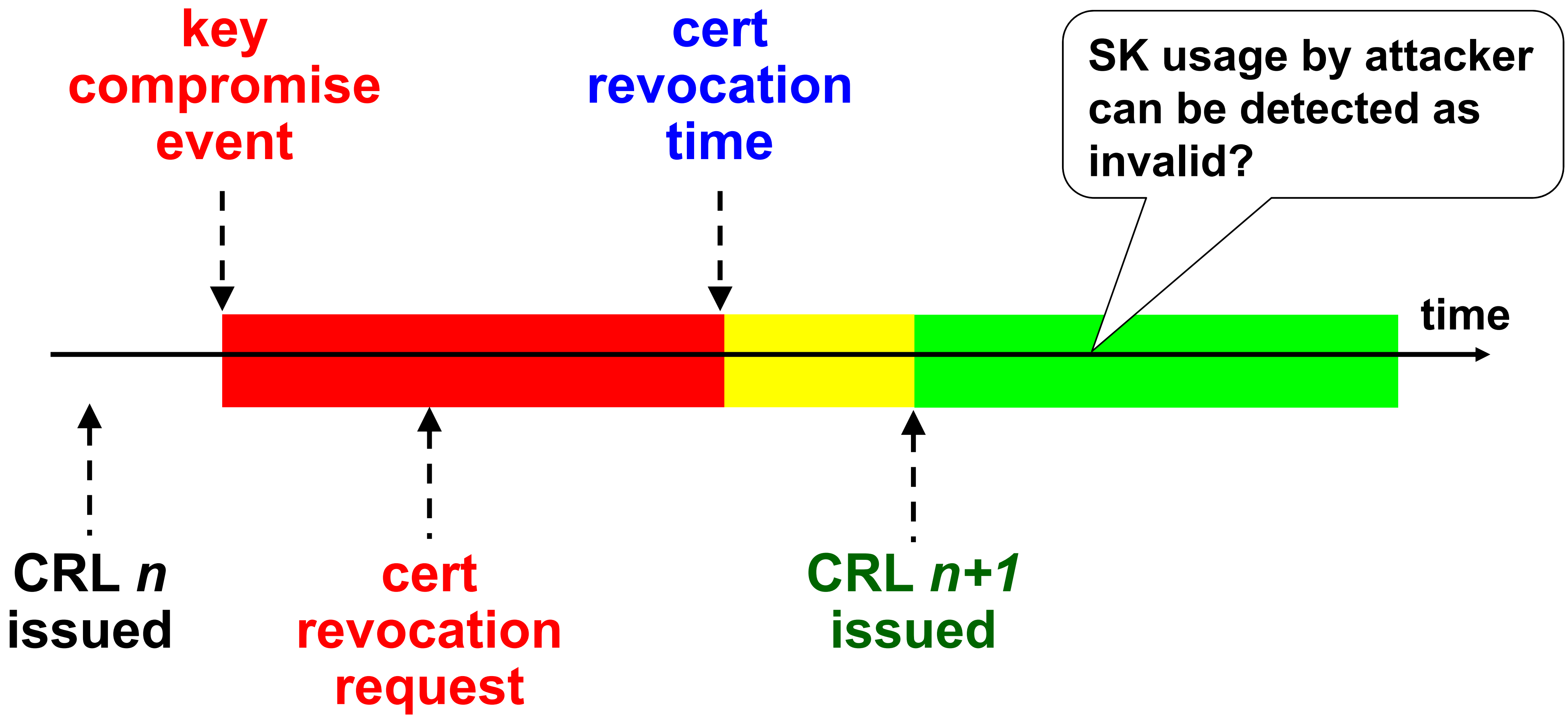
X.509 CRL version 2

```
CertificateList ::= SEQUENCE {
   tbsCertList            TBSCertList,
   signatureAlgorithm     AlgorithmIdentifier,
   signatureValue         BIT STRING }
TBSCertList ::= SEQUENCE {
   version                Version OPTIONAL,
                          -- if present, version must be v2
   signature              AlgorithmIdentifier,
   issuer                 Name,
   thisUpdate             Time,
   nextUpdate             Time OPTIONAL,
   revokedCertificates    SEQUENCE {
      userCertificate     CertificateSerialNumber,
      revocationDate      Time,
      crlEntryExtensions Extensions OPTIONAL
   } OPTIONAL,
   crlExtensions          [0] Extensions OPTIONAL
}
```

Extensions of CRLv2

- **crlEntryExtensions:**
  - reason code
  - hold instruction code
  - invalidity date
  - certificate issuer
- **crlExtensions:**
  - authority key identifier
  - issuer alternative name
  - CRL number
  - delta CRL indicator
  - issuing distribution point

Certificate revocation timeline



Efficient management of CRLs

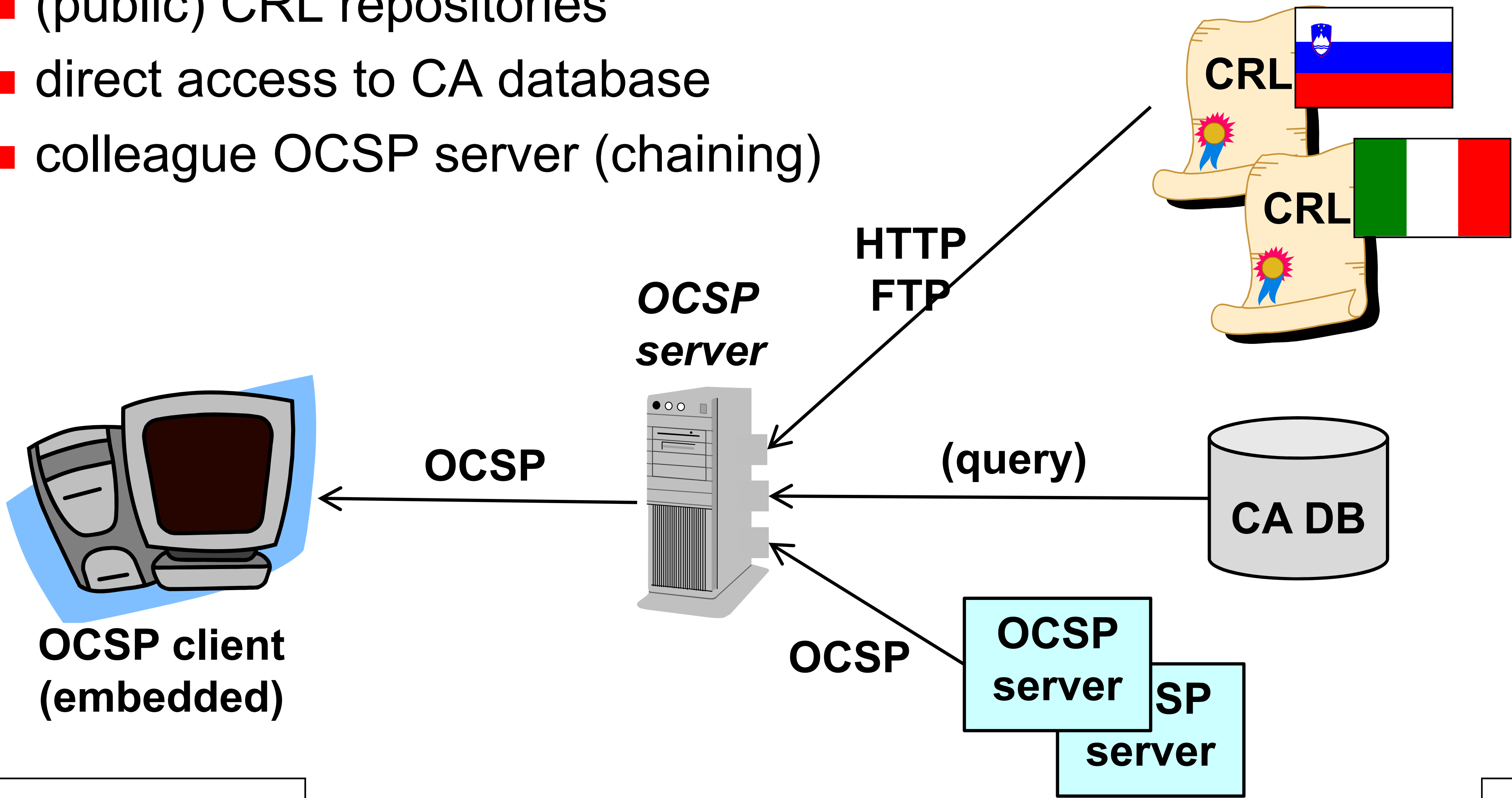
- **problem: the CRLs can become very big and consequently costly to download and to examine**
- **various solutions**
- **eliminate the revocation following the first CRL issued after the expiry date of the certificate**
  - you should keep an archive of past CRLs
- **publish complete CRL and then only the differences**
  - a base CRL (no. N) and then delta CRL (difference w.r.t. N)
  - fast download but burden to build the complete CRL
- **partition the CRL in groups (by properly using the CRLDP)**
  - cert with SN < 1000 : CRLDP= http://.../crl_1_1000.der
  - cert with 1000 < SN < 2000 : CRLDP=…/crl_1001_2000.der

OCSP

- **RFC-6960: On-line Certificate Status Protocol**
- **C/S protocol to verify if a certificate is valid NOW:**
  - good / revoked/ unknown
- **response digitally signed by the responder**
  - to avoid fake responses
- **the responder MAY accept only signed requests**
  - to restrict its clients
- **the signature certificate of the OCSP server cannot be verified with OCSP itself!**
- **OCSP implemented as**
  - binary protocol (no default port)
  - encapsulated in HTTP or HTTPS (but also LDAP, SMTP, …)

OCSP source of information

- **possible to obtain information from:**
  - (public) CRL repositories
  - direct access to CA database
  - colleague OCSP server (chaining)



OCSP – the protocol (I)

- **request:**
  - protocol version
  - target certificate identifier(s)
  - extensions (optional, MAY be processed or ignored)
- **response:**
  - version of the response syntax
  - name of the responder
  - responses for each certificate in the request
  - extensions (optional)
  - signature algorithm OID
  - signature computed across hash of the response

OCSP – the protocol (II)

- **in the request, each certificate is identified by its certID:**
  - hashAlgorithm (AlgorithmIdentifier)
  - issuerNameHash (hash of Issuer's DN)
  - issuerKeyHash (hash of Issuer's public key)
  - serialNumber (CertificateSerialNumber)
- **the response contains a SingleResponse for each cert in the request:**
  - certID
  - certStatus (good, revoked, unknown)
    - if revoked, revocationTime and revocation Reason (optional)
  - thisUpdate
  - nextUpdate (optional)

Models of OCSP responder

- **CA Responder**
  - OCSP server operated by the CA, which signs the responses with its own private key
  - risk of exposing the private key, so rarely adopted
- **Designated Responder (also Authorized Responder)**
  - the OCSP server signs the responses with different pairs key:cert, based on the CA for which it is responding
  - TTP operated / paid by the CA
- **Trusted Responder**
  - the OCSP server signs the responses with a pair key:cert independent of the CA for which it is responding
  - company responder or TTP paid by the users

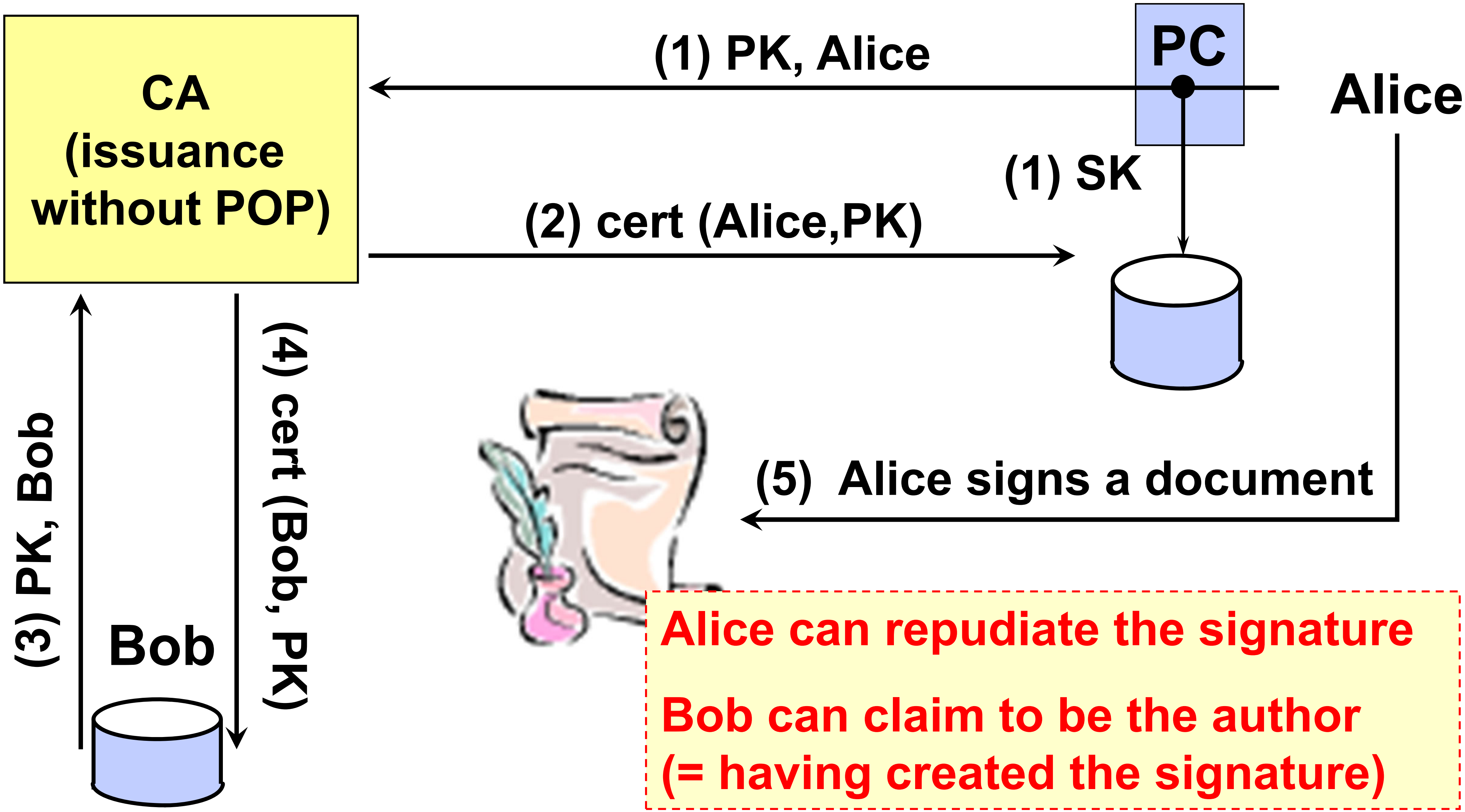
Attacks against OCSP

- **replay attack:**
  - (attack) copy response (valid certificate) and replay it after that certificate is revoked
  - (defence) client may insert a nonce (id-pkix-ocsp-nonce) in its request, to be included in the data signed by the server
- **DoS attack:**
  - (attack) flood the server with many requests, since each one requires a real-time on-line digital signature
  - (defence) pre-compute responses, with three timestamps:
    - thisUpdate, nextUpdate, producedAt
    - risky! open door for replay attacks, as a pre-computed response cannot contain a specific nonce

Proof-Of-Possession (POP)

- **POP = the CA has sufficient guarantees regarding the possession of the private key of an entity that requires a certificate (the Subject)**
- **issuance of certificates without POP allows various attacks**
  - different situation depending on the use of the key (POP fundamental to guarantee non-repudiation)
  - POP not always critical for encryption

Absence of POP – possible risks



Alice can repudiate the signature

Bob can claim to be the author (= having created the signature)

Countermeasures

- **best method: POP at signing time**
  - the signer inserts a reference to the certificate (e.g. a hash) among the signed information
  - thus, the signature value is a function of the certificate too (depends on it)
  - currently not supported by the security protocols
- **alternative solution: the CA issues a certificate only if it has the proof that the certificate requester owns the private key**

Methods for POP (I)

- **OOB methods**
  - keys generated by CA/RA and delivered in secure token (e.g. smart-card, USB crypto-token); it is considered as proof the possession of the token
  - policies of key-recovery/key-backup (very risky!!!): the CA maintains a copy of all private keys – how can it protect them efficiently?

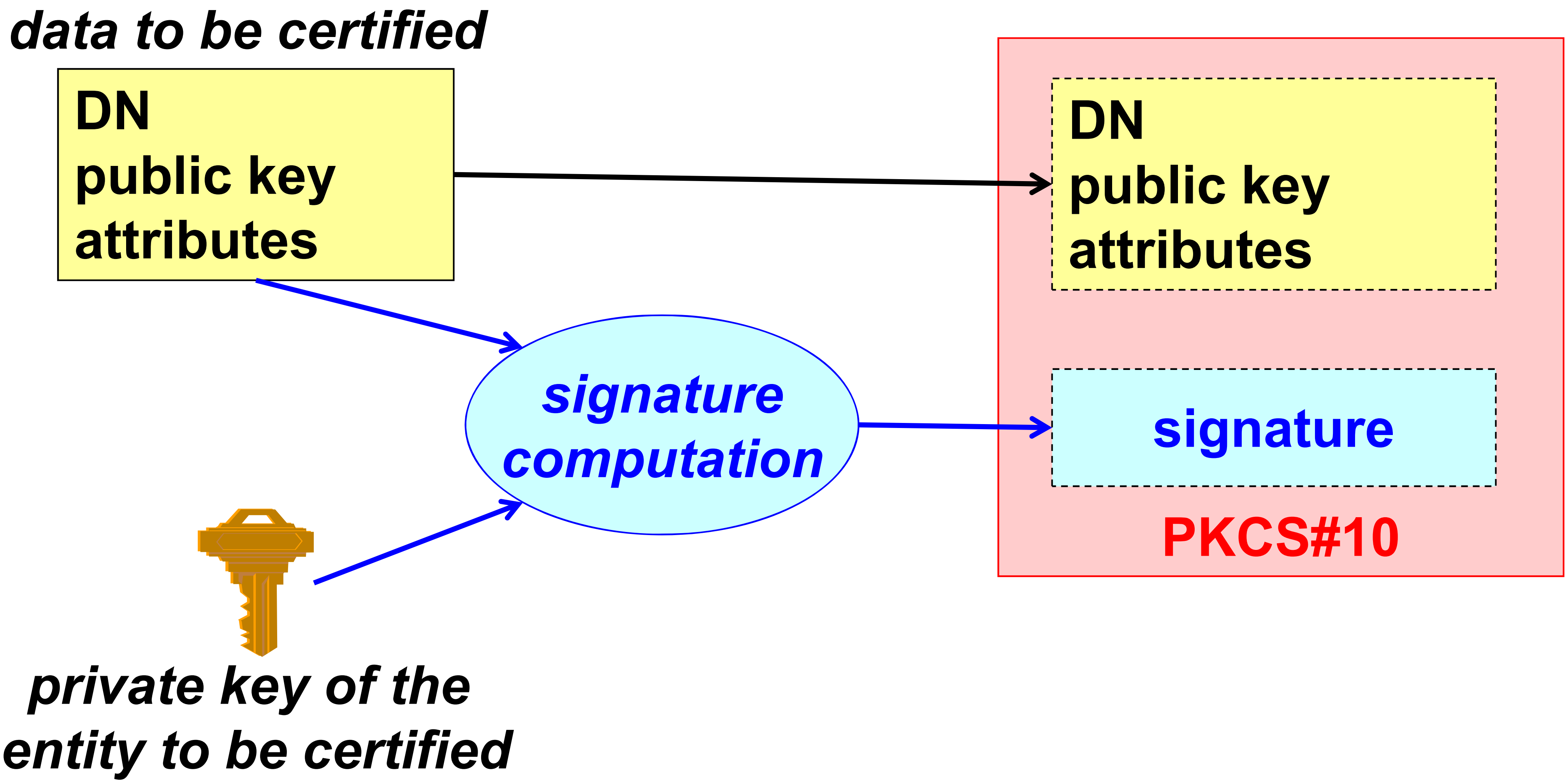
Methods for POP (II)

- **on-line methods**
  - for signing and encryption keys
    - use self-signed formats (PKCS#10, SPKAC): the CA verifies the signature and obtains POP
  - for encryption keys (no signature)
    - use of challenge-response protocols that involve a decryption operation (=use of the private key)
    - certificate returned in an encrypted form (subsequent revocation if the certificate is not used)

PKCS #10

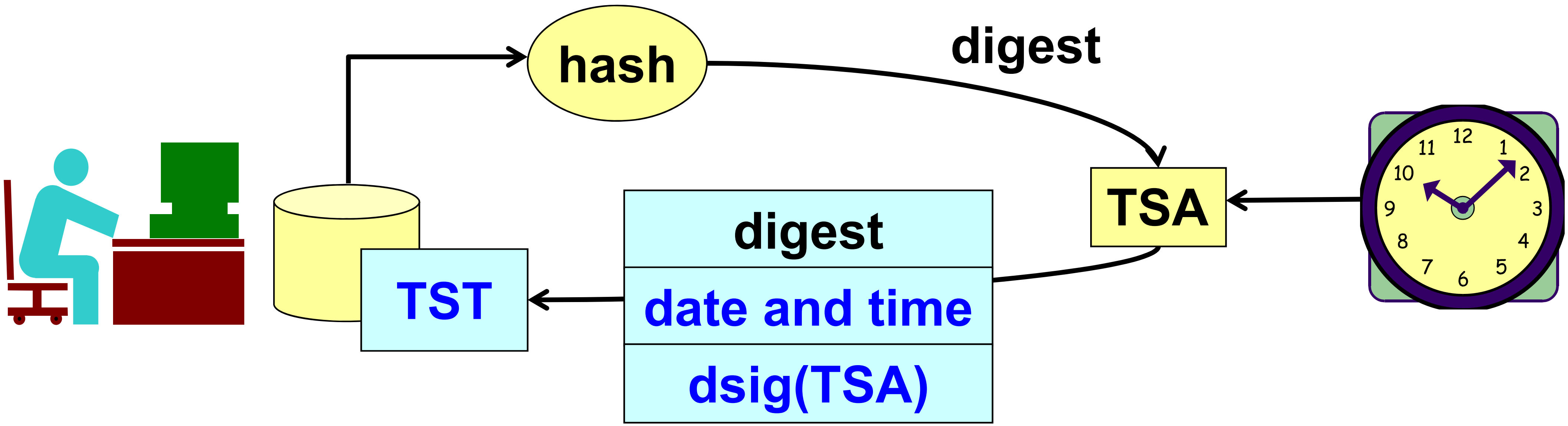
- **RFC-2986 = PKCS #10 (v 1.7)**
- **RFC-5967 = application/pkcs10 media type**
- **format for a certificate request**
  - aka CSR (Certificate Signing Request)
- **the request contains**
  DN + public key + (optional) attributes
- **possible attributes:**
  - "challenge password" (for registration / revocation)
  - attributes to be inserted in the certificate
    (e.g. those described in PKCS #9)
  - other information about the requestor

PKCS#10


data to be certified
DN
public key
attributes
signature
computation
private key of the
entity to be certified
DN
public key
attributes
signature
PKCS#10

Time-stamping

- **proof of creation of data before a certain point in time**
- **TSA (Time-Stamping Authority)**
- **RFC-3161:**
  - request protocol (TSP, Time-Stamp Protocol)
  - format of the proof (TST, Time-Stamp Token)

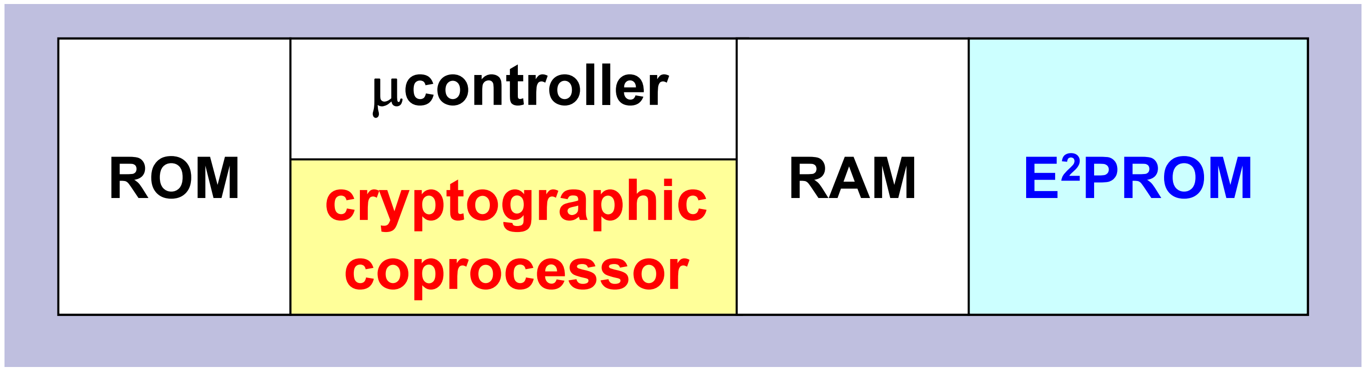


PSE (Personal Security Environment)

- **each user should protect:**
  - his own private key (secret!)
  - the certificates of the trusted root CAs (authentic!)
- **software PSE:**
  - (encrypted) file of the private key
- **hardware PSE:**
  - passive = protected keys (same as sw PSE)
  - active = protected keys + crypto operations
- **mobility is possible in both cases (but with problems)**

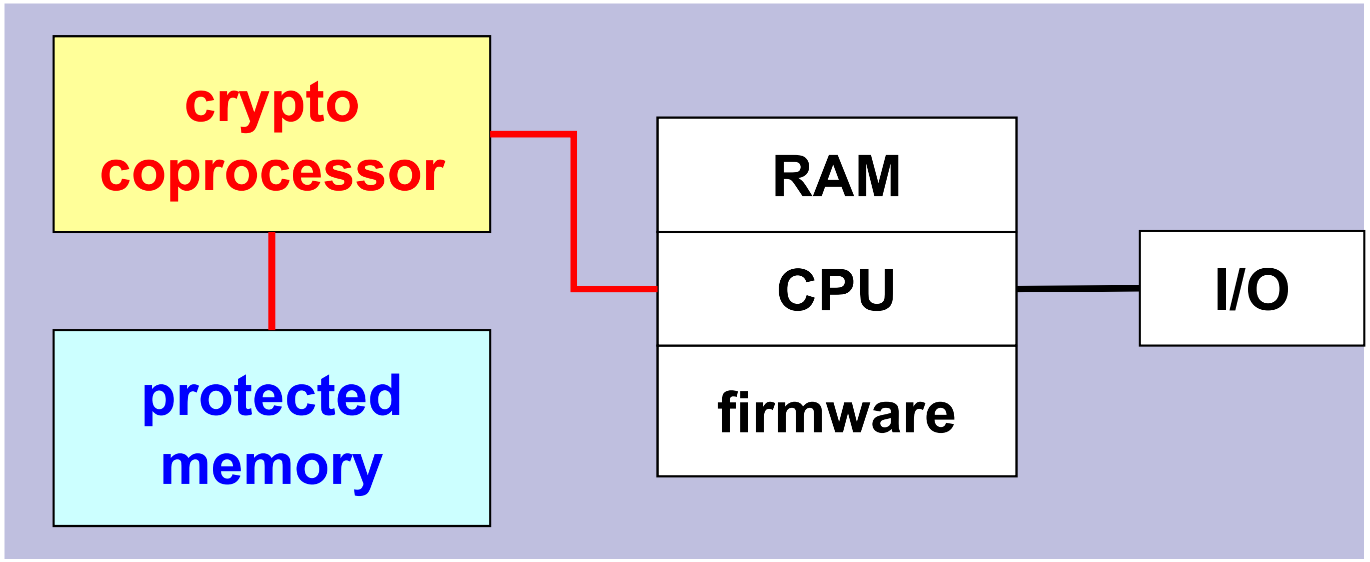
Cryptographic smart-card

- **chip cards with memory and/or autonomous cryptographic capacity**
- **simple: symmetric crypto (DES, AES, SHA)**
- **complex: asymmetric crypto (RSA, DSA, ECDSA)**
  - length of the key?
  - generation of the private key on board?
- **small memory (EEPROM): 4 - 64 kB**



HSM (HW Security Module)

- **cryptographic accelerator for servers**
  - secure storage of private key
  - autonomous encryption capabilities (RSA, sometimes symmetric algorithms too)
- **form factor: PCI board, or external device (USB, SCSI, …), or IP network device (netHSM)**

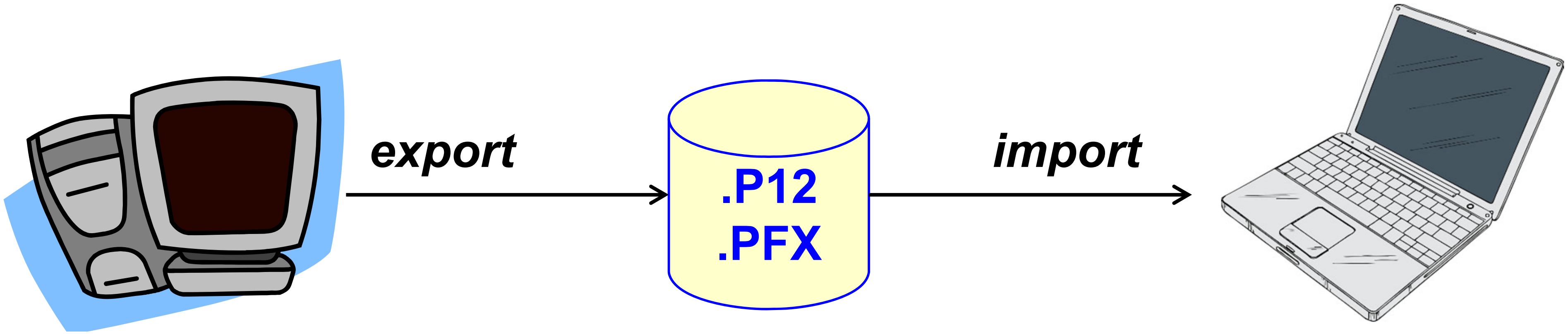


ISO 7816-x standards for smart-card

7816-1: Physical format
2: Contact characteristics
3: Electrical signals and protocols
4: Inter-industry commands
5: Application identifiers
6: Inter-industry data elements
7: SCQL (SC Query Language)
8: Inter-industry security commands
9: Commands for card management
10: Electronic signals and answer to reset for sync cards
11: Personal verification through biometric methods
12: Cards with contacts — USB electrical interface and operating procedures
13: Commands for application management in multi-application environment
15: Cryptographic information application

PKCS#12 format (security bag)

- RFC-7292, transport of (personal) cryptographic material among applications / different systems
- transports a private key and one or more certificates
- transports the digital identity of a user
- used by Java, Microsoft, Mozilla, Google, …
- criticized from the technical point of view (especially in the MS implementation) but widely used



How-to display a X.509 certificate

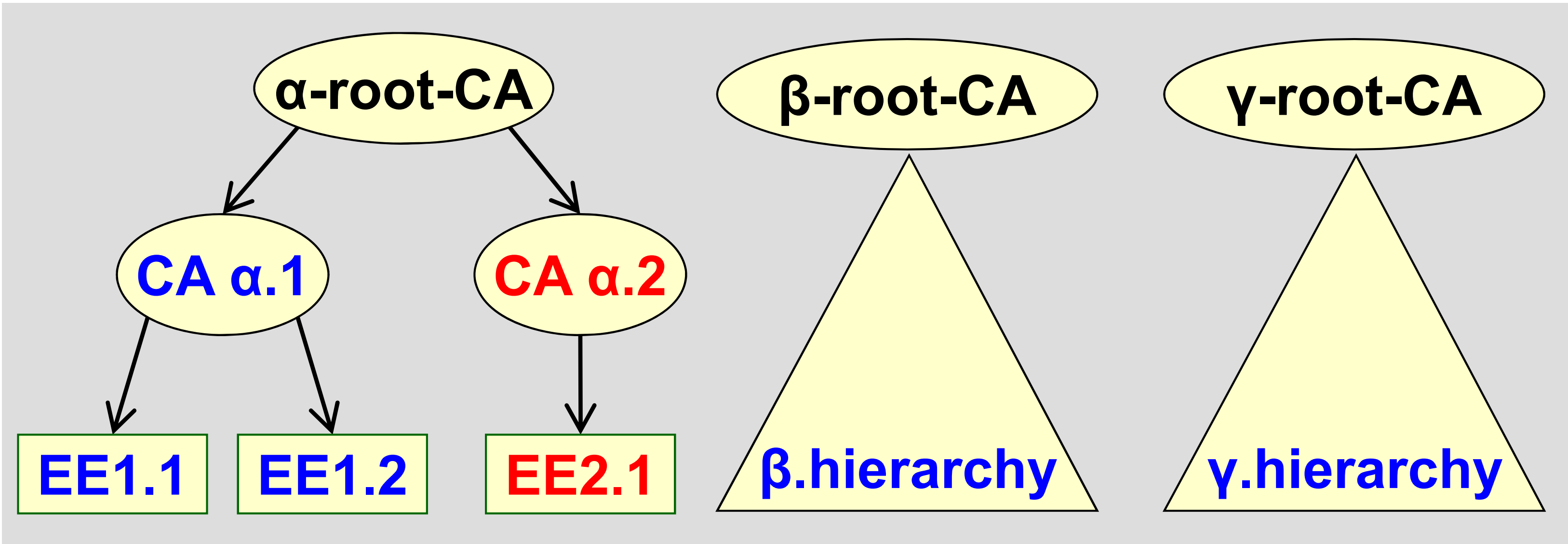
- **openssl asn1parse:**
  - display the actual ASN.1 structure
  - may convert from the input format (PEM or DER) to DER
- **openssl x509:**
  - signs / verifies a certificate
  - “-text” displays the content of the certificate
- **dumpasn1:**
  - displays the content of the certificate
- **NOTE:**
  - DER (Distinguished Encoding Rules) is a binary encoding of ASN.1 (it’s a subset of BER, Basic Encoding Rules)
  - PEM is the *armoured base64* encoding of DER

openssl asn1parse / x509

`-in F`	input file
`-inform X`	input format (DER, PEM)
`-out F`	output file (only DER)
`-i`	indent the text output
`-noout`	do not provide the PKC as output
`-offset N`	begin the analysis from byte N
`-length N`	consider only the first N bytes
`-dump`	hexadecimal dump of the unknown data
`-oid F`	file containing supplemental OIDs
`-text`	generate a readable text version of a PKC
`-strparse O`	interpret the blob at offset O

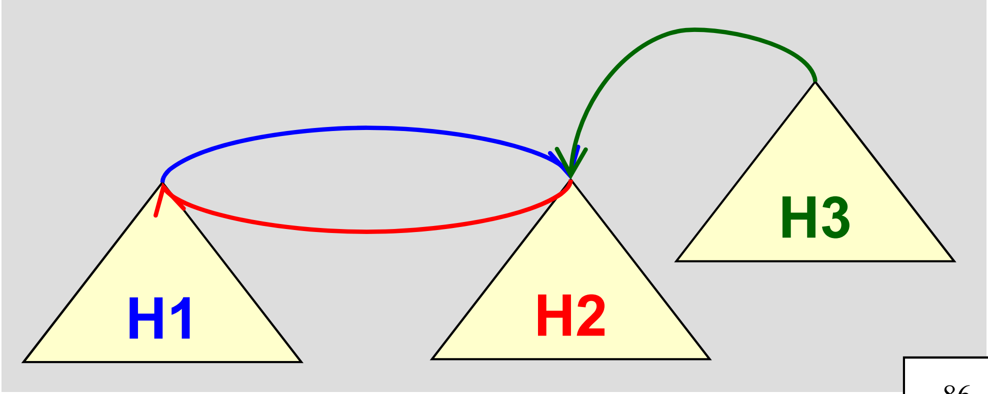
Hierarchical PKI

- a tree rooted at a self-signed **root CA**
- very easy to build a certification path between any two EEs
- natively supported by all applications
- legal / commercial / political problems have created not a single hierarchy spanning the world but rather a **forest** (i.e. many separate trees)



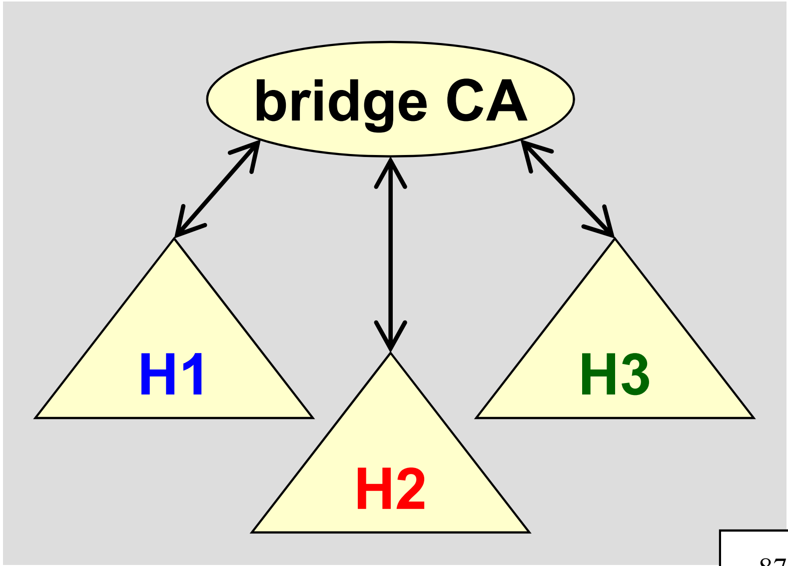
Mesh PKI

- **two hierarchical PKIs may unilaterally or bilaterally trust each other by issuing a cross-certificate**
  - a certificate issued by root CA for another root CA
- **… but cross-certificates are not automatically recognized by standard applications**
  - which certificate chain should the application consider?
- **for complete trust among all hierarchies, we need N(N-1)/2 cross-certificates**
- **rarely used in practice (mostly due to the application problem)**



Bridge PKI

- **to simplify management operation (addition/deletion of a CA) and trust transitivity, a bridge CA is introduced**
  - it is trusted and cross-certified with each root CA
  - does not certify any other CA or EE
- **complete trust is created with just N cross-certificates**
- **… but this is not automatically recognized by standard applications**
- **most notable example:**
  - US federal PKI
    - https://fpki.idmanagement.gov/



What to do with PKC mistakenly issued or issued by compromised CA ?

- **difficult for domain owners to detect certificates fraudulently issued for their domain (servers)**
  - browsers aren't good at detecting **rapidly** malicious websites if they receive (e.g. in a TLS connection):
    - mistakenly issued certificates
    - certificates issued by a compromised CA
  - when these situations occur, it takes time to detect the problem, revoke the certificates, make the browser aware
  - until then … browser users think that they are visiting an authentic site and/or the connection is secure
    - fake server
    - MITM attacks

Mistakenly issued certificates: some examples

- **(2011) an intruder managed to issue itself a valid certificate for the domain google.com and its subdomains from the prominent Dutch Certificate Authority DigiNotar**
  - certificate issued in July 2011 but may have been used maliciously for weeks before the detection on August 28, 2011, of large-scale MITM attacks on multiple users in Iran
- **(2011) the Comodo Group suffered from an attack which resulted in the issuance of nine fraudulent certificates for domains owned by Google, Yahoo!, Skype, and others**
- **(July 2014) An unknown number of mis-issued certificates were issued by a sub-CA of India CCA, the Indian NIC. Due to the scope of the incident, the sub-CA was wholly revoked, and India CCA was constrained to a subset of India's ccTLD namespace**

HTTP Key Pinning (HPKP)

- **HTTPS site specifies the digest of its own public key and/or one or more CAs in its chain (but the root!)**
- **UA caches the key and refuses connecting to a site with a different key**
  - it's a TOFU (Trust On First Use) technique
  - dangerous when loosing control of the key
  - problems with key updates
    - always include at least one backup key
- **URI to report violations (enforcing or report-only mode)**
- **protection against fake web site trying to impersonate the real site by using fraudulently obtained PKC**
- **deprecated in favour of Certificate Transparency**

Certificate Transparency (CT)

- **https://www.certificate-transparency.org/**
- **an open, global auditing and monitoring system**
  - based on public log of issued certificates
  - allows domain owners to verify that no fraudulent certificates have been issued for their domains
- **experimental protocol originally proposed by Google and standardized by the IETF Public Notary Transparency WG**
- **issuance and existence of TLS certificates open to analysis by domain owners, CA, and domain users**
  - web clients should only accept certificates publicly logged
  - it should be impossible for a CA to issue a certificate for a domain without it being publicly visible

Certificate Transparency – main ideas

- **make it impossible (or very difficult) for a CA to issue a PKC for a domain without making it visible to the domain owner**
- **provide an open auditing and monitoring system that lets any domain owner or CA determine whether certificates have been mistakenly or maliciously issued**
- **protect users from being provisioned by certificates that were mistakenly or maliciously issued**
- **CT framework is composed of several actors:**
  - Submitter
  - Loggers
  - Monitor
  - Auditors

CT – Log servers

- **are the core of the CT system**
- **servers that maintain a secure log of TLS certificates**
  - append-only
    - certificates can only be added to a log (cannot be deleted, modified, or retroactively inserted into a log)
  - cryptographically assured
    - protected by Merkle Tree Hashes to prevent tampering and misbehaviour
  - publicly auditable
    - anyone can query a log (via HTTPS GET and POST) and verify that it's well behaved, or verify that a TLS certificate has been legitimately appended to the log

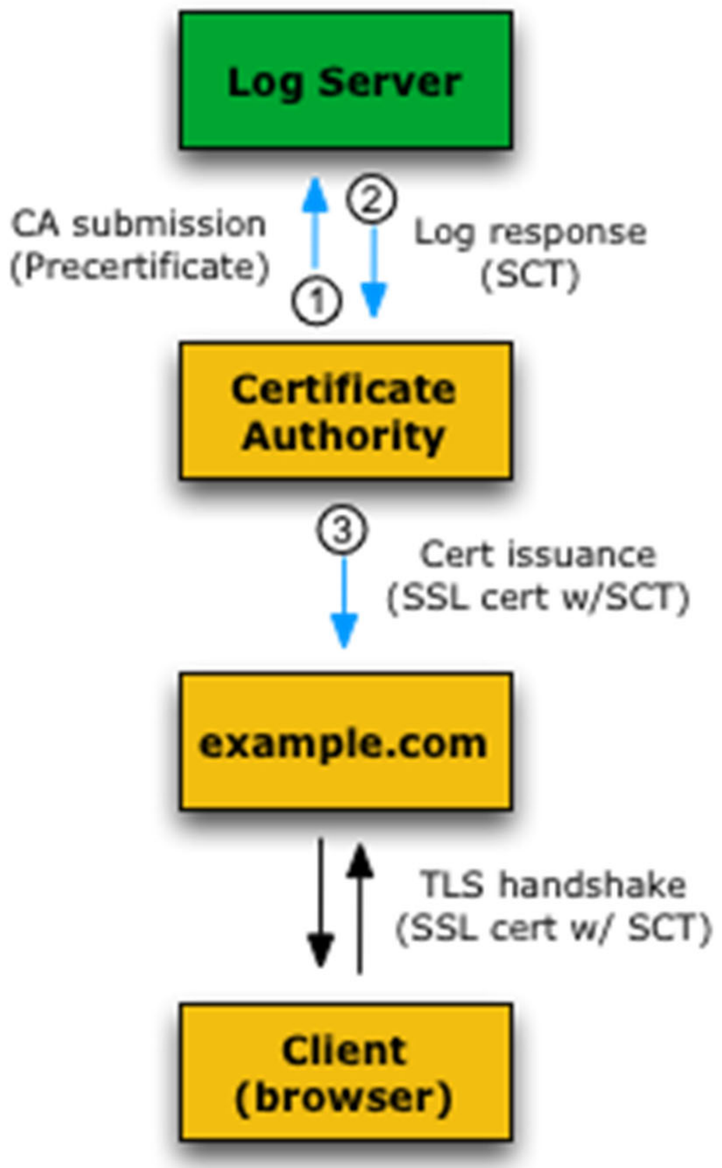
CT – log signature and support

- **logs must be digitally signed**
  - (v1.0) NIST P256 or RSA-2048
  - (v2.0) NIST P-256, Deterministic ECDSA, or Ed25519
- **Deterministic ECDSA?**
  - RFC-6979 “Deterministic Usage of the DSA and ECDSA”
  - if the K value used in computation is not random, SK can be computed from signature (!) – problem in embedded systems
- **CT support in Chrome/Chromium/Safari:**
  - 1 SCT from a currently approved log
  - duration < 180 days: 2 SCTs from once-approved logs
  - duration > 180 days: 3 SCTs from once-approved logs
- **CT support in Firefox: pending implementation**

CT – operations

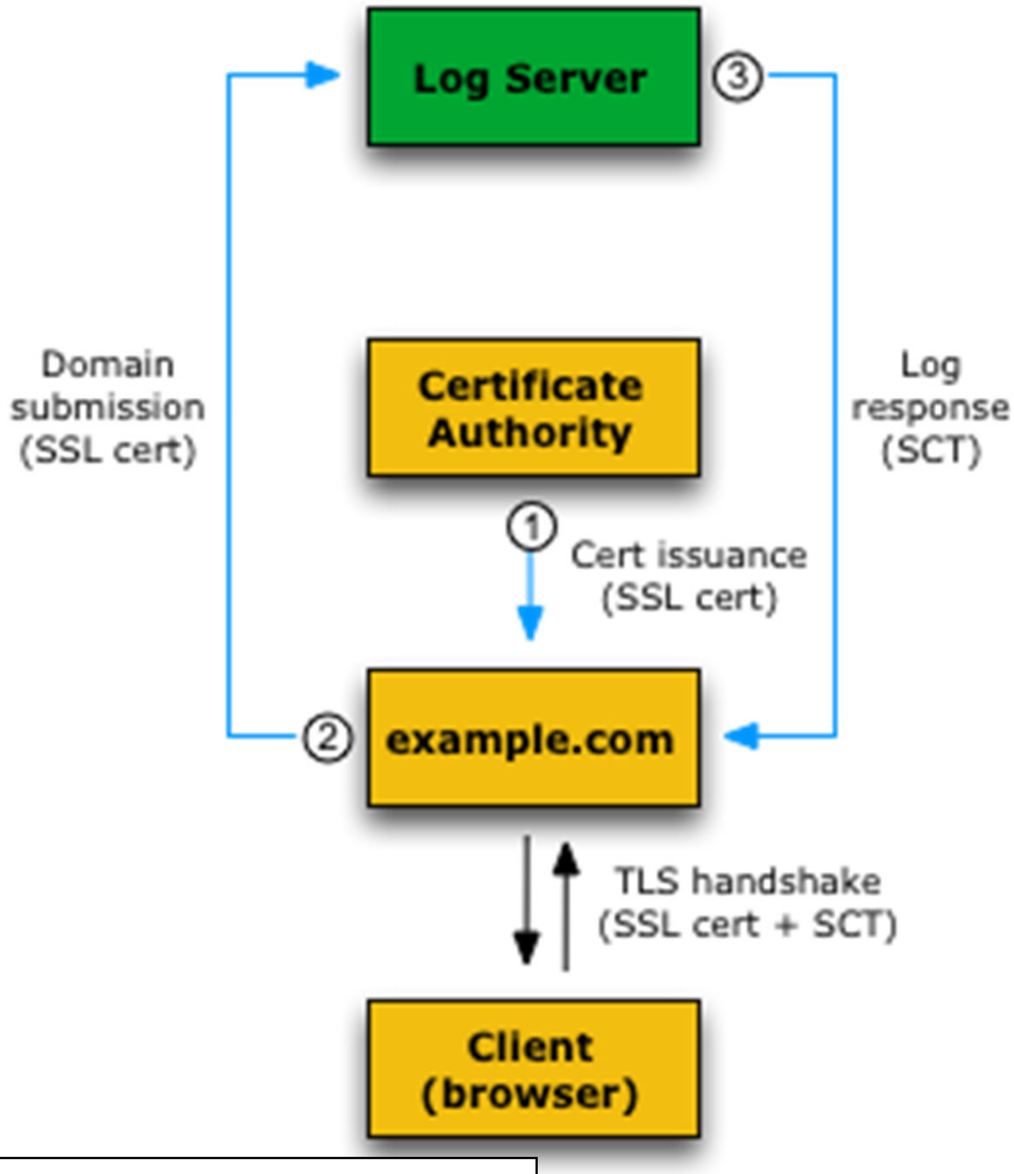
- **anyone can submit a certificate to a log server, although most certificates will be submitted by CAs and server operators**
- **the logger provides each submitter with a promise to log the certificate within a certain amount of time**
  - the promise is called a Signed Certificate Time-stamp (SCT)
- **the SCT accompanies the certificate throughout its lifetime**
- **how is the SCT (created by the Log servers) delivered with the certificate (created by the CA)?**
  - X.509v3 extension
  - TLS extension
  - OCSP Stapling

SCT via X.509v3 extension



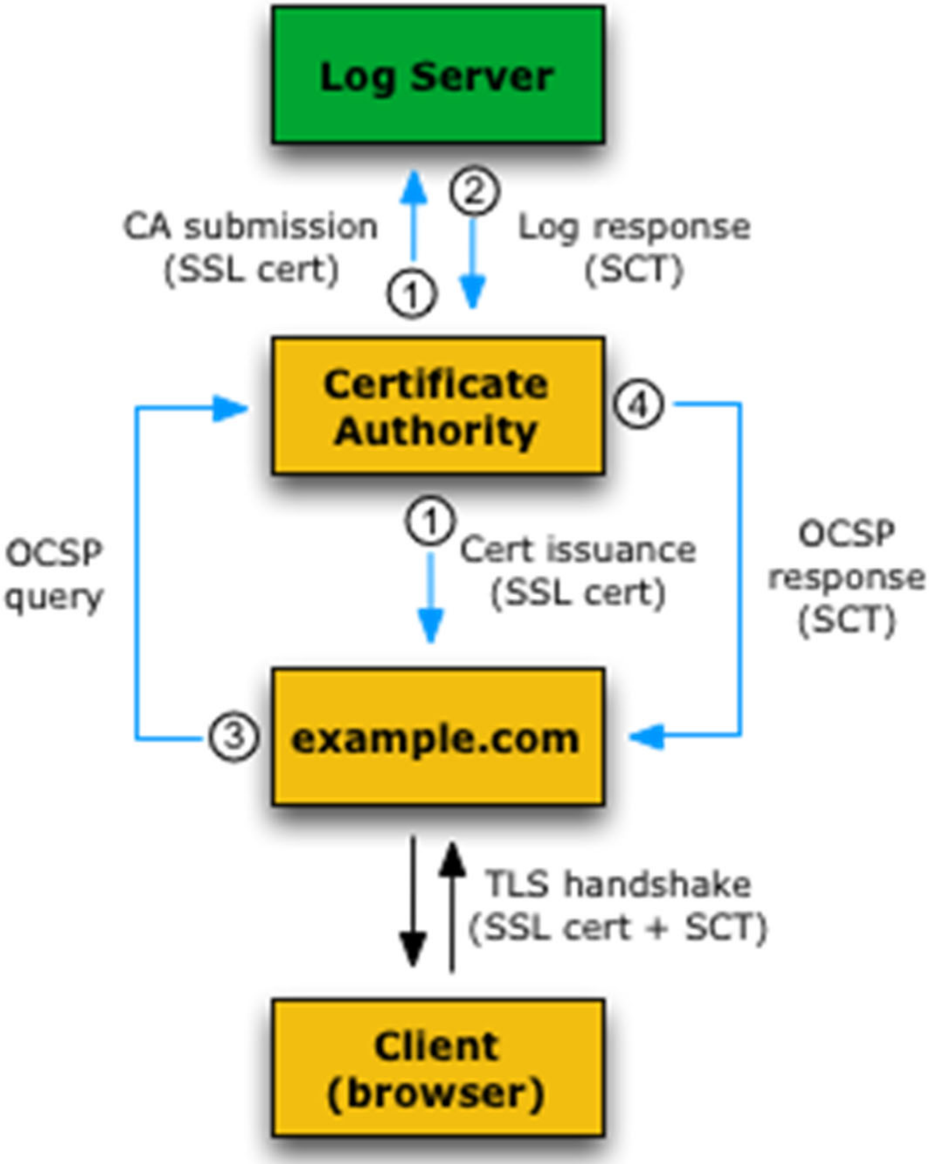
- **the CA submits a pre-certificate to the log server**
- **the log server returns an SCT**
- **the CA then attaches the SCT to the pre-certificate as an X.509v3 extension, signs the certificate, and delivers the certificate to the server operator**

SCT via TLS extension



- the CA issues the certificate to the server operator
- the server operator submits the certificate to the log server
- the log server sends the SCT to the server operator
- server uses the TLS extension signed_certificate_timestamp to deliver the SCT to the client during the TLS handshake

SCT via OCSP stapling



- the CA provides the certificate to the log server and the server operator
- the server operator makes an OCSP query to the CA
- the OCSP response contains the SCT, which the server can include in the OCSP extension during the TLS handshake

CT – Submitter and Monitors

- **Submitters**
  - submit certificates (or partially completed certificates) to a log server and receive a SCT
- **Certificate Monitors**
  - public or private services that watch for misbehaving logs or suspicious certificates
  - periodically contact and download information from log servers
  - inspect every new entry, keep copies of the entire log and verify the consistency between published revisions of the log

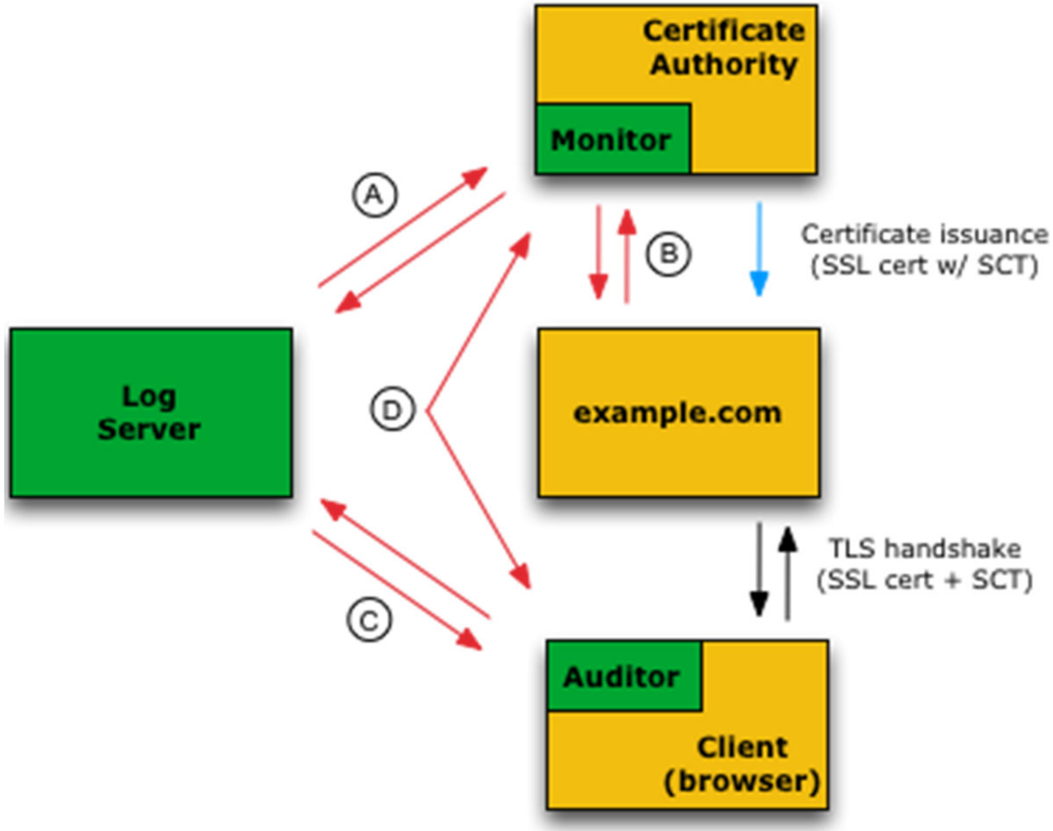
CT – Certificate Auditors

- **Certificate Auditors**
  - lightweight software components that perform two functions
    - verify overall integrity of logs
      - periodically fetch and verify log proofs
      - log proof = signed cryptographic hash a log uses to prove it's in good standing
    - verify that a particular certificate appears in a log
      - the CT framework requires that all TLS certificates be registered into a log
      - if a certificate has not been registered in a log, it's a sign that the certificate is suspect, and TLS clients may refuse to connect to sites that have suspect certificates

CT – Auditors

- If a TLS client determines (via auditor) that a certificate is not in log, it can use the SCT from the log as evidence that the log has not behaved correctly
- auditors and monitors exchange information about logs through a gossip protocol

A possible CT system configuration (I)



- (A) Monitors watch logs for suspicious certs and verify that all logged certs are visible
- (B) Cert owners query monitors to verify there are no illegitimate certs logged for their domain
- (C) Auditors verify that logs are behaving properly and can also verify that a particular cert has been logged
- (D) Monitors and auditors exchange info about logs to detect forked / branched logs

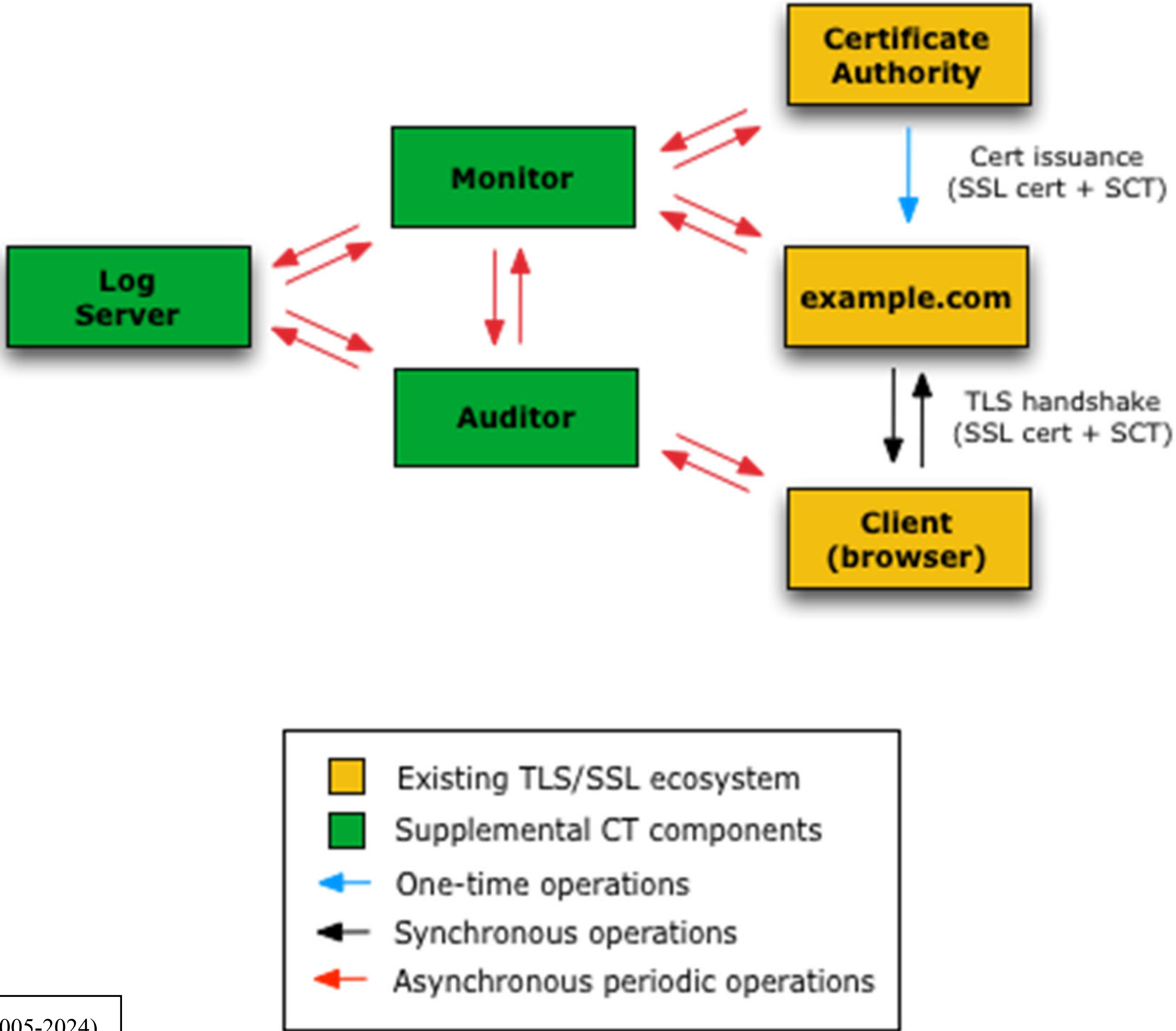
A possible CT system configuration (II)

- **CT doesn't prescribe any particular configuration**
- **a possible configuration:**
  - CA obtains an SCT from a log server and incorporates it into the TLS certificate using an X.509v3 extension
  - CA issues the certificate (with the SCT attached) to the server
  - during the TLS handshake, the TLS client receives the TLS certificate (and thus also the SCT)
  - TLS client validates the certificate and validates the log signature on the SCT to verify that the SCT was issued by a valid log and that the SCT was actually issued for that certificate
    - if discrepancies are found, the TLS client may reject the certificate

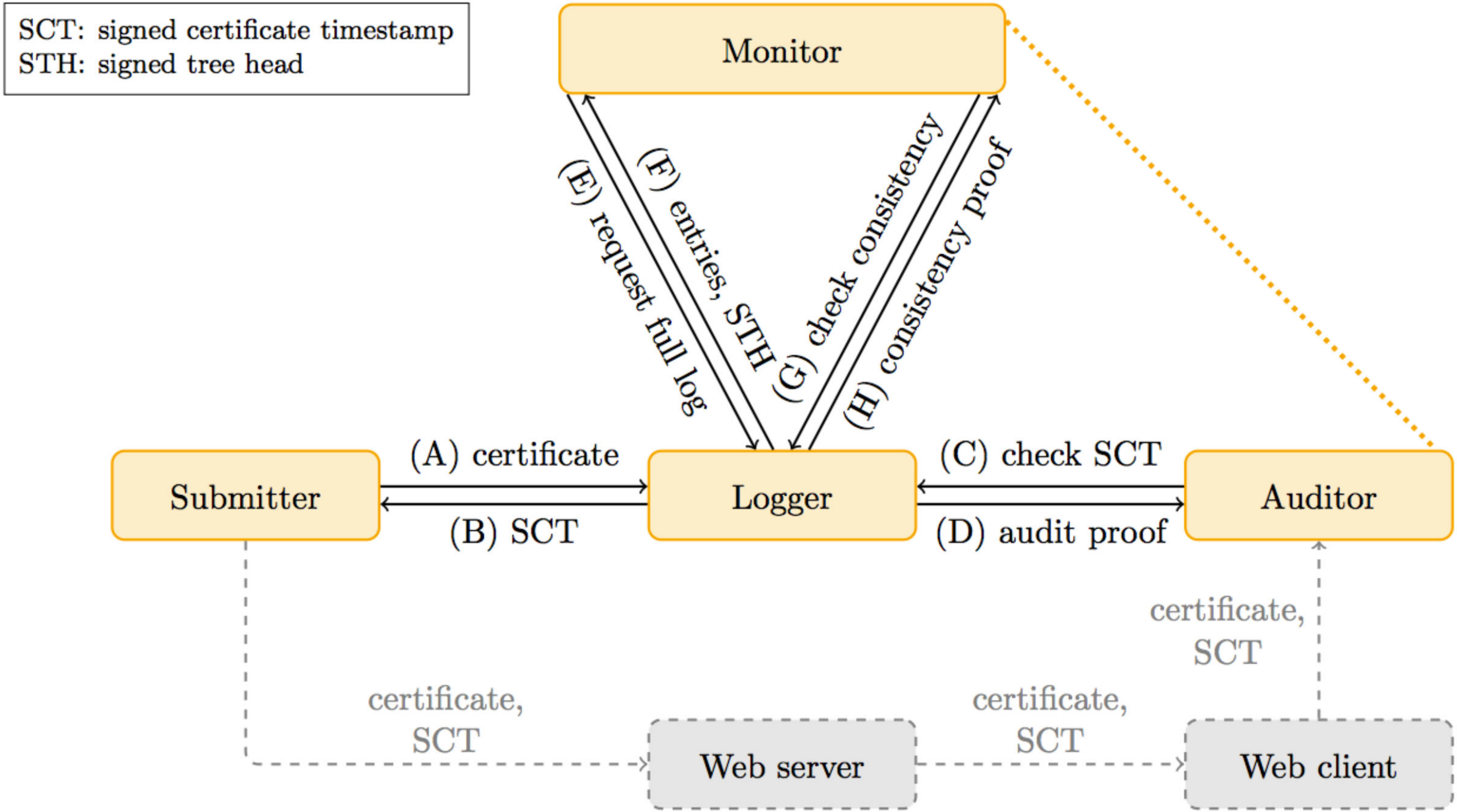
A possible CT system configuration (III)

- **monitors operated by the CA**
- **auditors built into the browser**
  - browser periodically sends a batch of SCTs to its integrated auditing component and asks whether the SCTs have been legitimately added to a log
  - auditor asynchronously contact the logs and perform the verification
- **monitors and auditors operate as standalone entities, providing paid or unpaid services to CAs and server operators**

Another possible CT system configuration



Interaction between entities in CT



CT – current log servers

- **it's dependent on the browser**
- **(October 2023)**
  - 6 valid log servers
    https://certificate.transparency.dev/logs/
    - CloudFlare, DigiCert, Google, Let’s Encrypt, Sectigo, TrustAsia
  - 10 public monitor
    https://certificate.transparency.dev/monitors/
    - Censys, CloudFlare, crt.sh (by Sectigo), DigiCert, Entrust, Facebook, KEYTOS, Hardenize, sslmate, ReportURI
    - many only for the domains served by themselves (e.g. CloudFlare)

ACME Protocol

- **RFC-8555 (mar-2019) "Automated Certificate Management Environment (ACME)"**
  - protocol for PKC management between EE and CA
- **created by the ISRG (Internet Security Research Group) for their CA service, Let’s Encrypt (letsencrypt.org)**
  - to foster TLS adoption … FREE OF CHARGE!
- **it allows PKC to be automatically requested and issued, without human interaction**
- **ACME interactions are based on exchanging JSON documents over HTTPS connections**

ACME protocol: details

- **ACME agent (client) installed on a web server proves to the CA (Let’s Encrypt) that the server controls a domain**
- **once the CA verifies this fact, it allows the client to request and install certificates at will**
- **the client can be programmed to perform certificate operations at fixed intervals**
- **no need to manually generate individual PKCS#10 requests, prove domain ownership, download, and configure the server certificate using ad-hoc procedures**
- **there are over 100 open-source ACME client**
  - https://letsencrypt.org/docs/client-options/
  - Certbot (from Let’s Encrypt), GetSSL, Posh-ACME, Caddy, ACMESharp, ...

How to exploit ACME?

- creation of an account at Let’s Encrypt
- domain validation
- issuance of certificate
- revocation of certificate

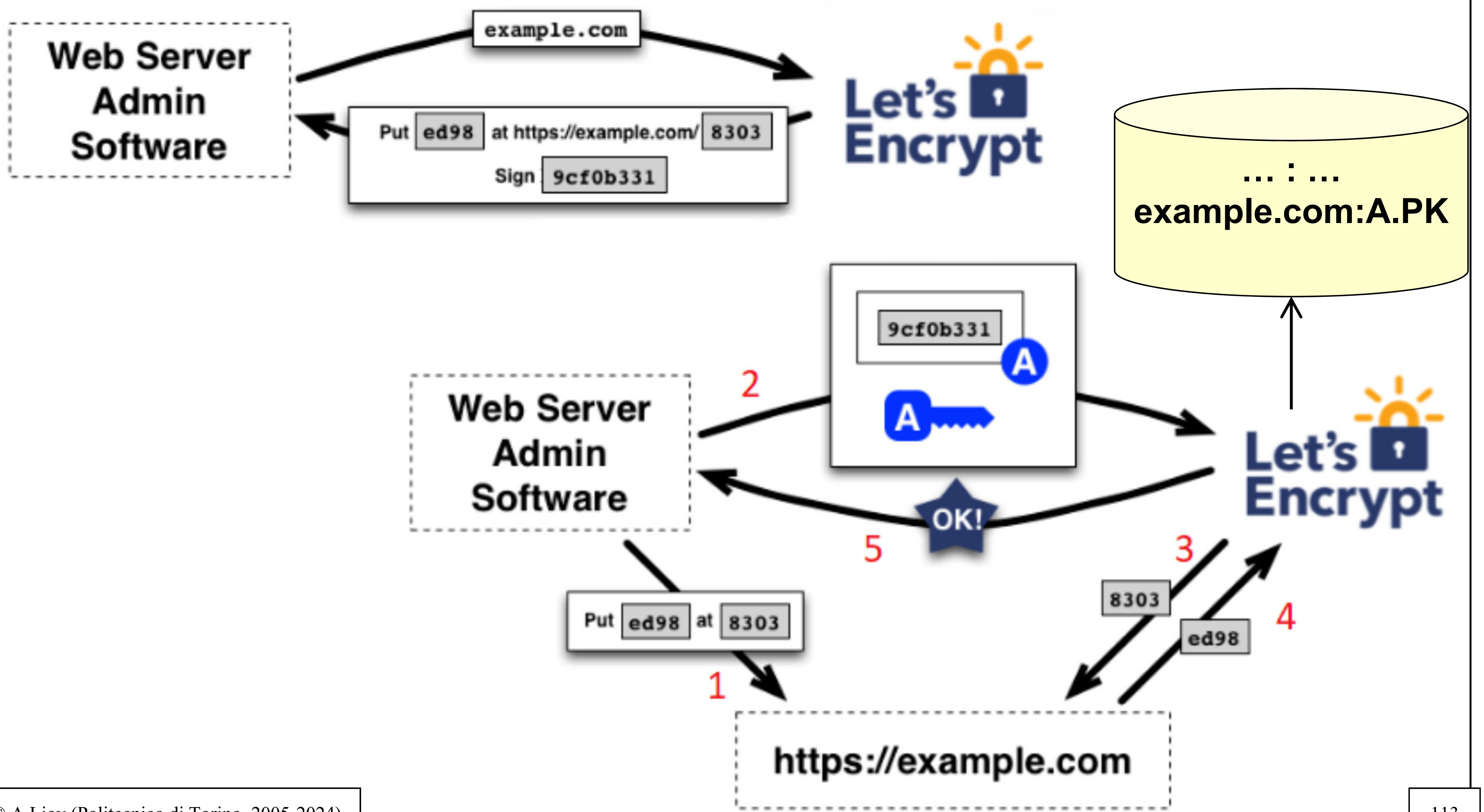
ACME – account creation

- need to be performed only once, before asking issuance / revocation of domain certificate(s)
- ACME client generates a private/public key pair used for authorization purpose, the "authorized key pair"
- the authorized PK is associated to the account registered at Let's Encrypt (LE)
- the authorized SK will be used to sign the certificate requests, while the authorized PK will be used by LE to validate the received certificate requests
- a single account is associated to one or more domains

ACME – domain validation

- ACME client must prove to the CA the control of the domain for which it will request certificates
- ACME client claims ownership of a domain
- CA challenges the client to store a value at a specific path inside the domain (step 1) or in a DNS record
- … and to sign a nonce with the authorized SK
- when the challenge is solved, the client sends the signed nonce to the CA to start the validation process (step 2)
- the CA downloads the response from the domain and verifies its correctness (steps 3 & 4) and the signature's one
- if the everything is correct, the CA approves domain ownership (step 5) and the client is enabled to request certificates for that domain

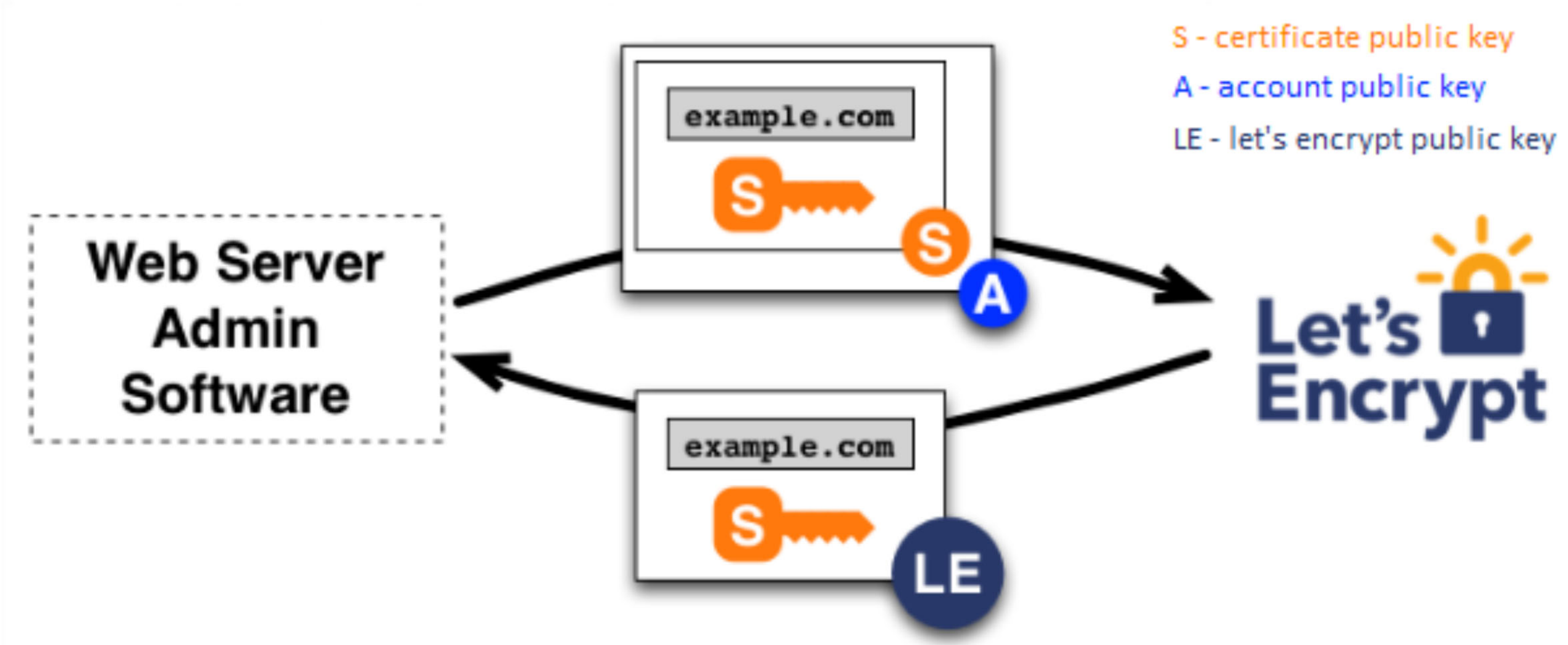
ACME – domain validation



ACME – certificate request and issuing

- **client uses the authorized key pair to request, renew, and revoke certificates for the domain**
- **the client constructs a PKCS#10 CSR and asks the CA to issue a certificate with a specified public key**
  - the CSR includes a signature by the SK corresponding to the PK in the CSR
  - the client also signs the whole CSR with the authorized SK for that domain

ACME – certificate request and issuing



ACME – certificate revocation

- client signs a revocation request with the authorized SK for the corresponding domain
- the CA verifies that the key is authorized
- if so, it revokes the certificate and includes this information in the proper revocation mechanisms (CRL and/or OCSP)

ACME – certificate revocation