BTI 4202: Centralized public-key infrastructures

Christian Grothoff

Berner Fachhochschule

24.5.2026

Learning Objectives

Public Key Infrastructures

Trust Issues in X.509

X.509 CA Alternatives

Digital Timestamping

Trust Agility

Part I: Public Key Infrastructures

Public Key Infrastructure (PKI)

Core problems:

- How can a user/application verify, if the private/public key pair belongs to the claimed person or entity (e.g. a web-server)?
- How does an application find the correct public key of a certificate issuer?

Public Key Infrastructure (PKI)

Core problems:

- How can a user/application verify, if the private/public key pair belongs to the claimed person or entity (e.g. a web-server)?
- How does an application find the correct public key of a certificate issuer?

Additional problems:

- How do we keep private keys safe?
- How do we distribute information if someone lost control over their private key?

A PKI allows us to recognize which *public key belongs to whom*, which *manages certification*, and establishes *trust*.

Private vs. Public PKI

Private PKI:

- Created, operated and maintained limited to private organization only.
- Issues certificates for internal purpose only. No trust relationship to external components, except cross-link to other private PKI.
- Free in terms of PKI structure and content of certificates.
- Use-cases: for User-Auth, VPN or Enterprise WPA (RADIUS)

Public PKI:

- Well known, reachable, and trusted on a more global basis.
- Operated and maintained by dedicated companies or by governmental services.
- Per default known by popular Internet applications.
- Example: X.509-Certificates for TLS, ZertES-based user certificates

The fairy tale of a global PKI

- Since the beginning of the asymmetric cryptography era the ultimate "dream" has been to establish a global PKI
- Because "everybody" trusts the national post offices, one can communicate securely with anyone else in the world.

The fairy tale of a global PKI

- Since the beginning of the asymmetric cryptography era the ultimate "dream" has been to establish a global PKI
- Because "everybody" trusts the national post offices, one can communicate securely with anyone else in the world.
- The inconvenient truth:
  - There is no global PKI today, and (hopefully) there never will be!
  - It is even hard to build a national PKI, accepted by a majority of application providers in one specific country!

The PKI Reality

Any PKI faces *authority* and *trust* issues:

- Who is authorized to issue a specific certificate?
  - Which are the CA's that claim authority to assign keys to names?
  - What makes them authoritative in respect to these names?
- How do they vet subjects?
- How are keys managed (by CA and subjects)?

A hierarchical trust model (trust in a forest of authorities) is still the "best" solution we have today.

PKI Components: Terminology

End Entity (EE) Denotes denote end-users, devices (e.g., servers, routers), or any other entity that can be identified in the subject field of a public key certificate. End entities typically consume PKI-related services.

Certification Authority (CA) Issue of certificates and (usually) CRLs. In practice, often integrates administrative functions of RAs and VAs.

Registration Authority (RA) Assumes administrative functions from the CA, primarily EE registration. Can assist in related areas (e.g. EE authentication, token distribution, key generation, revocation reporting, archiving of key pairs).

Validation Authority (VA) Responsible for providing information on whether certificates are actually valid or not.

Repository Generic term used to denote any method for storing certificates and CRLs so that they can be retrieved by EEs.

Certificate Authorities (CAs)

CAs are the foundation of the PKI since they are the only components that can issue certificates.

- CAs act as a trust anchors, and are thus high-security relevant components. Their private keys must be kept in a particularly secure environment.
- In a hierarchical system, a Root CA signs its own certificate with its private key (self-signed) according to a specific procedure.
- To update its own self-signed key a Root CA uses a *key-rollover* procedure to renew Root CA's key pair.

Baseline Requirements (BR)

In several statements a CA describes the practices employed to support its certification services. For this purpose the IETF published templates in the form of a common RFC [3]:

- **Certificate Practice Statement (CPS)**: a statement of the practices which a certification authority employs in issuing certificates.
- **Certificate Policy (CP)**: a named set of rules that indicates the applicability of a certificate to a particular community and/or class of application with common security requirements.

The CAB-Forum extended the requirements of a CA with the release of "Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates":

`https://cabforum.org/baseline-requirements`

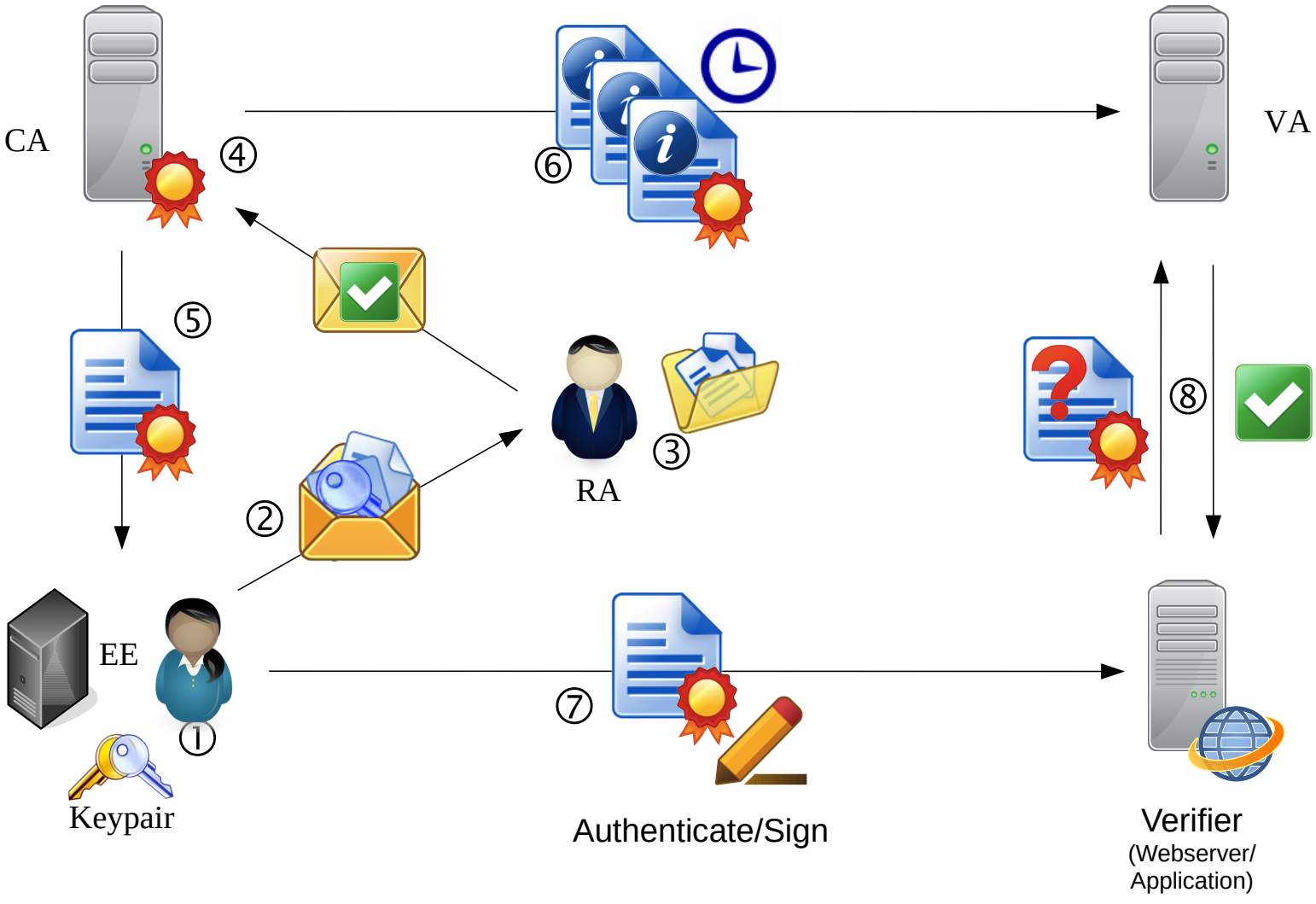
Registration Authorities (RAs)

- Roughly spoken: a registration location is where a user has to be registered personally.
- Registration Authorities convert “real” identities into “digital” identities.
- A Registration Authority needs to have experience with unfailing identification of users.

Examples:

- Bank / post office
- Small meshed network of local agencies
- Human resources / personal office in a company
- IT service department

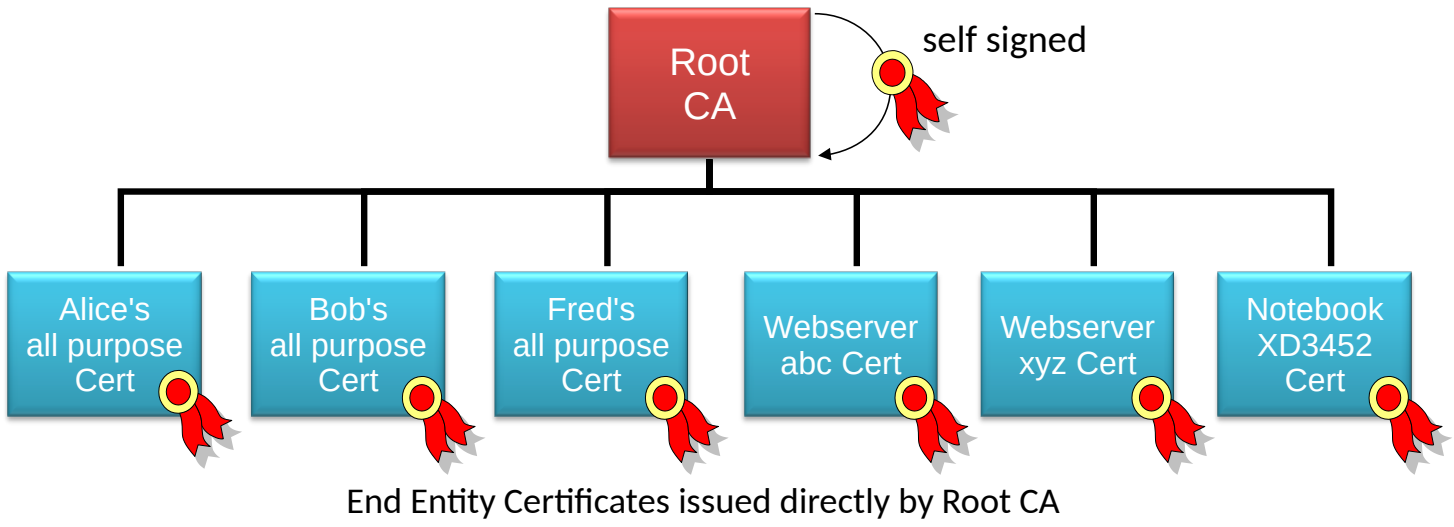
Idealized Issuance and Validation Process



Single-Level CA

- The Root CA signs directly certificates of EE
- Difficult to achieve different key policies $\rightarrow$ typically issues all-purpose certificates (encryption, digital signature, authentication)

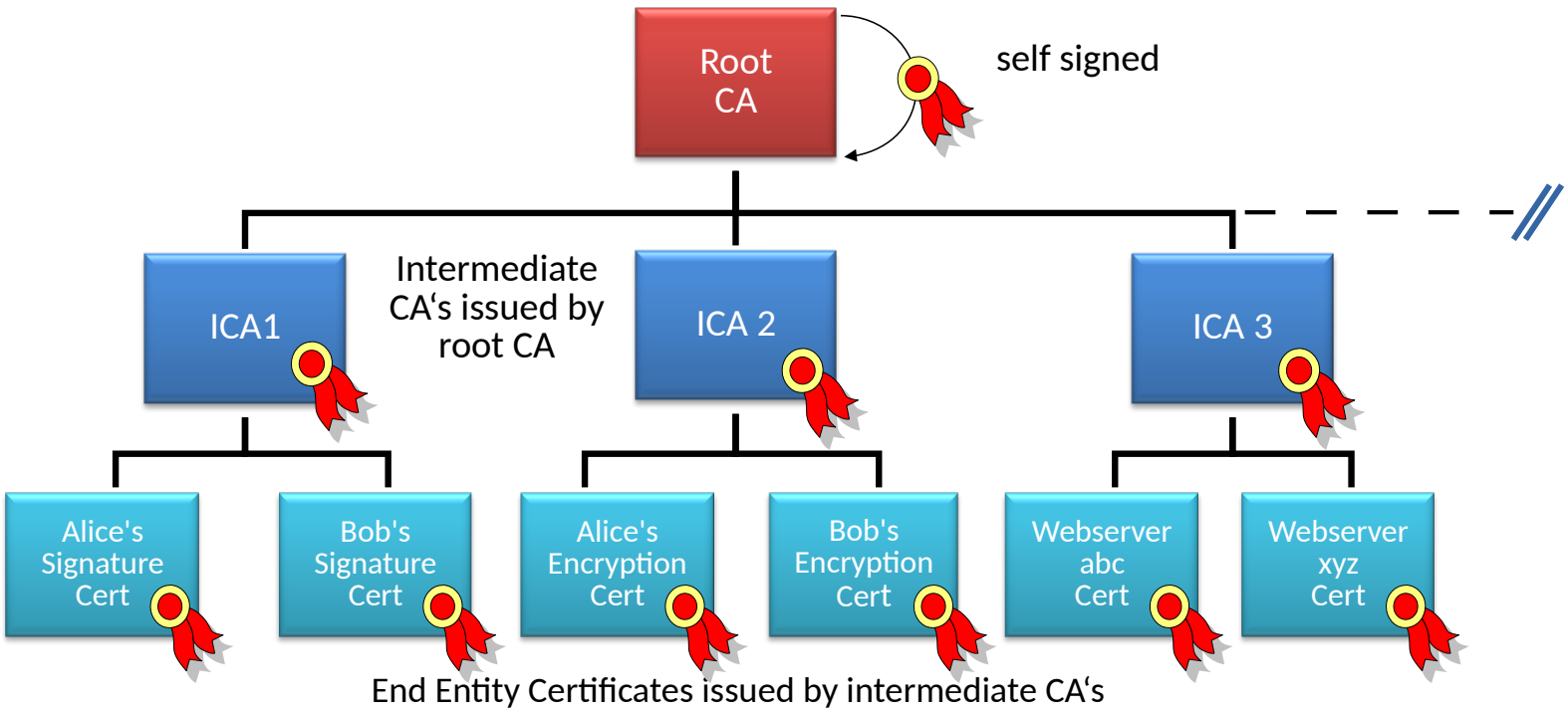
$\Rightarrow$ **Suitable for private CAs**



Multi-Level CA

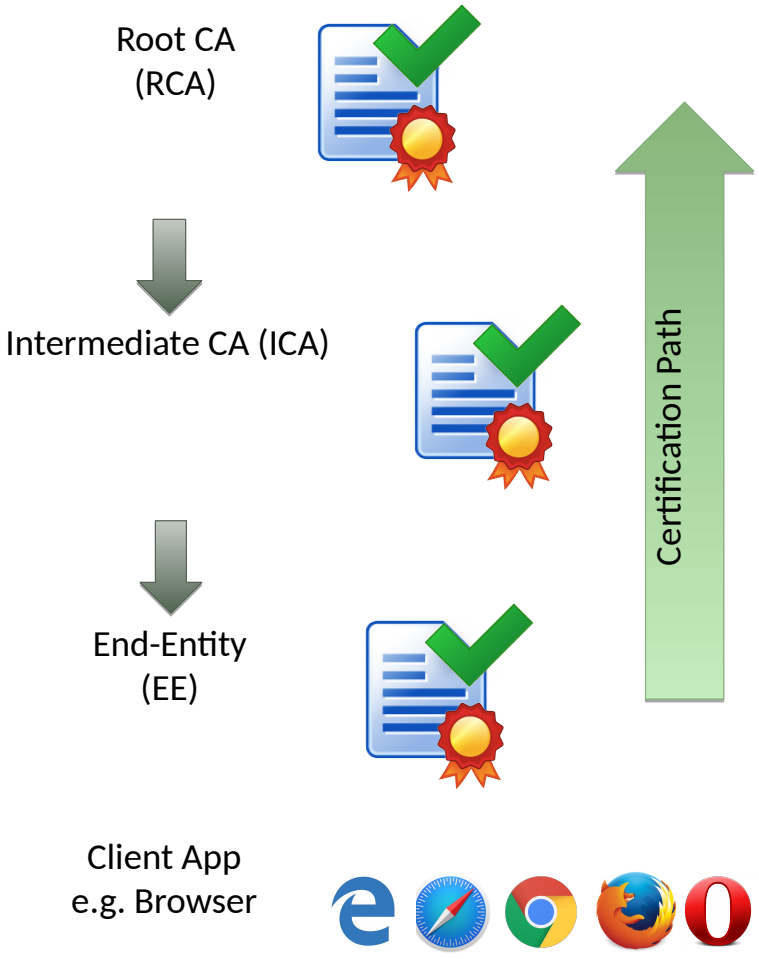
- Multi-Level PKIs use a tree model with the root CA at the top and Intermediate CAs (SubCAs, Signing CAs, Issuing CAs) at intermediate levels.
- RootCA typically only signs ICA certificates and is in general turned off and kept in safe environment.
- The SubCAs are signing and issuing EE certificates.
- Enables possibility to force specific key policies by issuing certificates for restricted purposes only:
  - Authentication (personal or machine certificates)
  - Data Encryption
  - Digital Signatures

Multi-Level CA



- ICA1 issues user certificates for digital signature only
- ICA2 issues user certificates for data encryption only
- ICA3 issues certificates for Web servers only
- ICAx issues ...

Hierarchical Certificate Verification Process



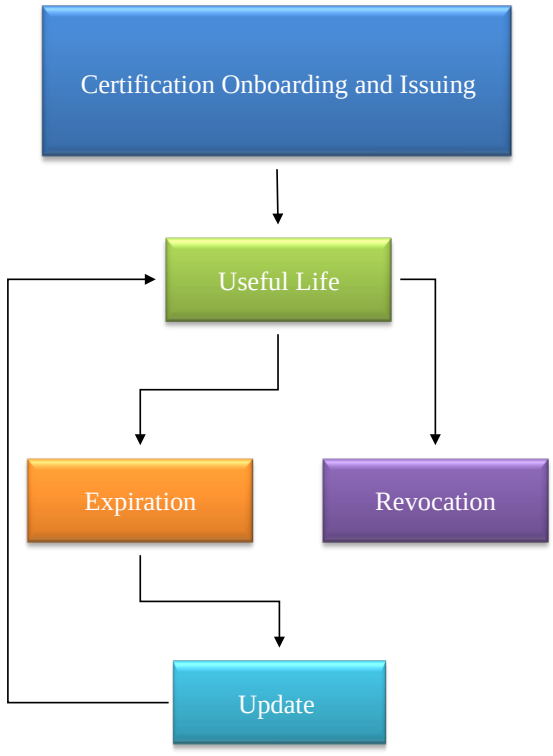
- RCA explicitly trusted
- Delegates trust to ICA
- ICA binds EE to key
- Client checks delegation chain

Certification Path Validation

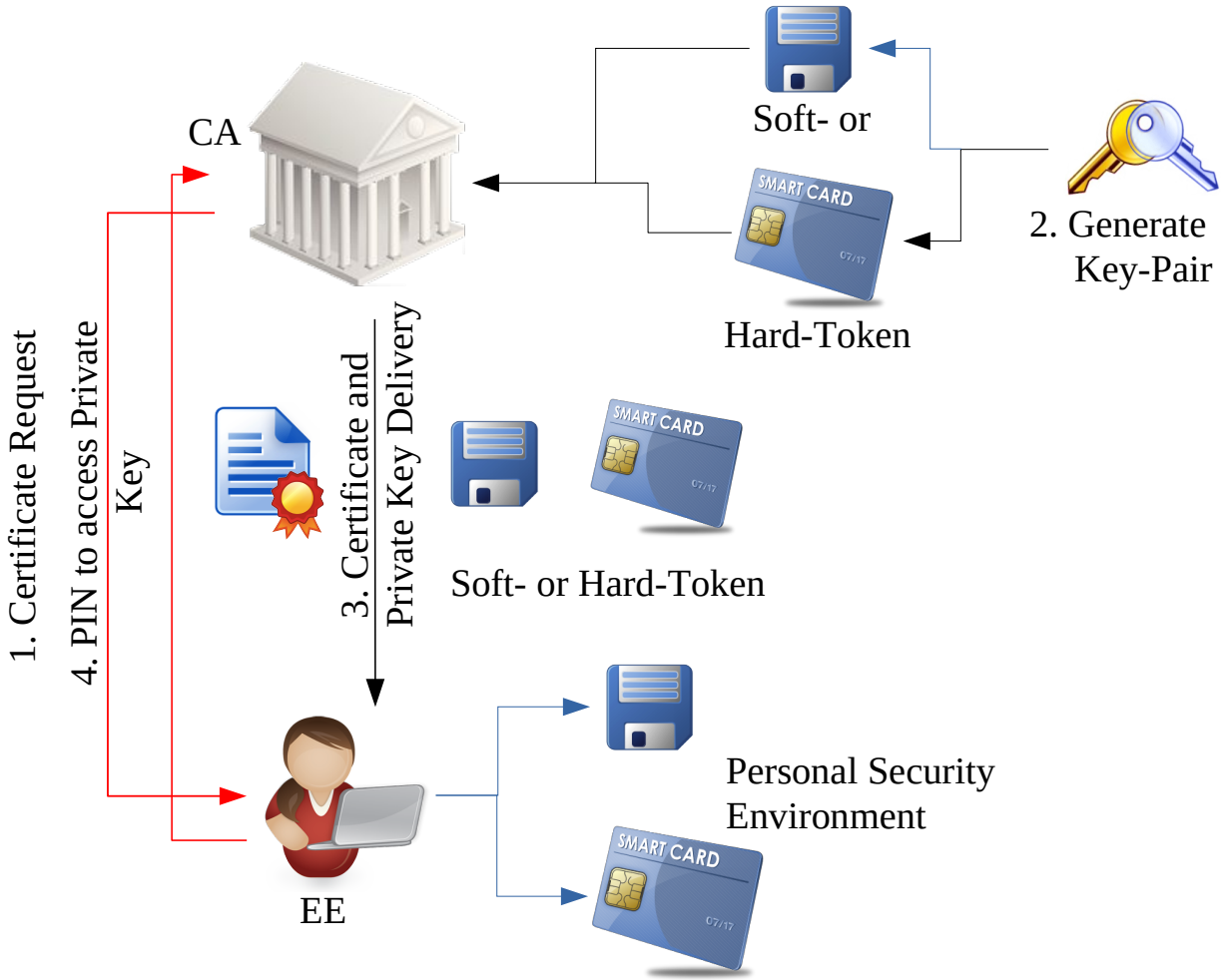
[4] describes the algorithm for validating certificates, certification path and certification policies. This gives an answer to the questions:

- Is a certificate signed by a recognized trust anchor (is the certification path rooted in a trusted CA)
- Can the digital signature on the certificate be properly verified?
- Is the certificate within its established validity period?
- Has the certificate not been revoked?
- Is the certificate being used in a manner that is consistent with the certificate policies, name constraints, key usage, etc.?

The Certificate Lifecycle



Key Generation by Certificate Authority (CA)



$\Rightarrow$ May allow key escrow/backup by issuing CA

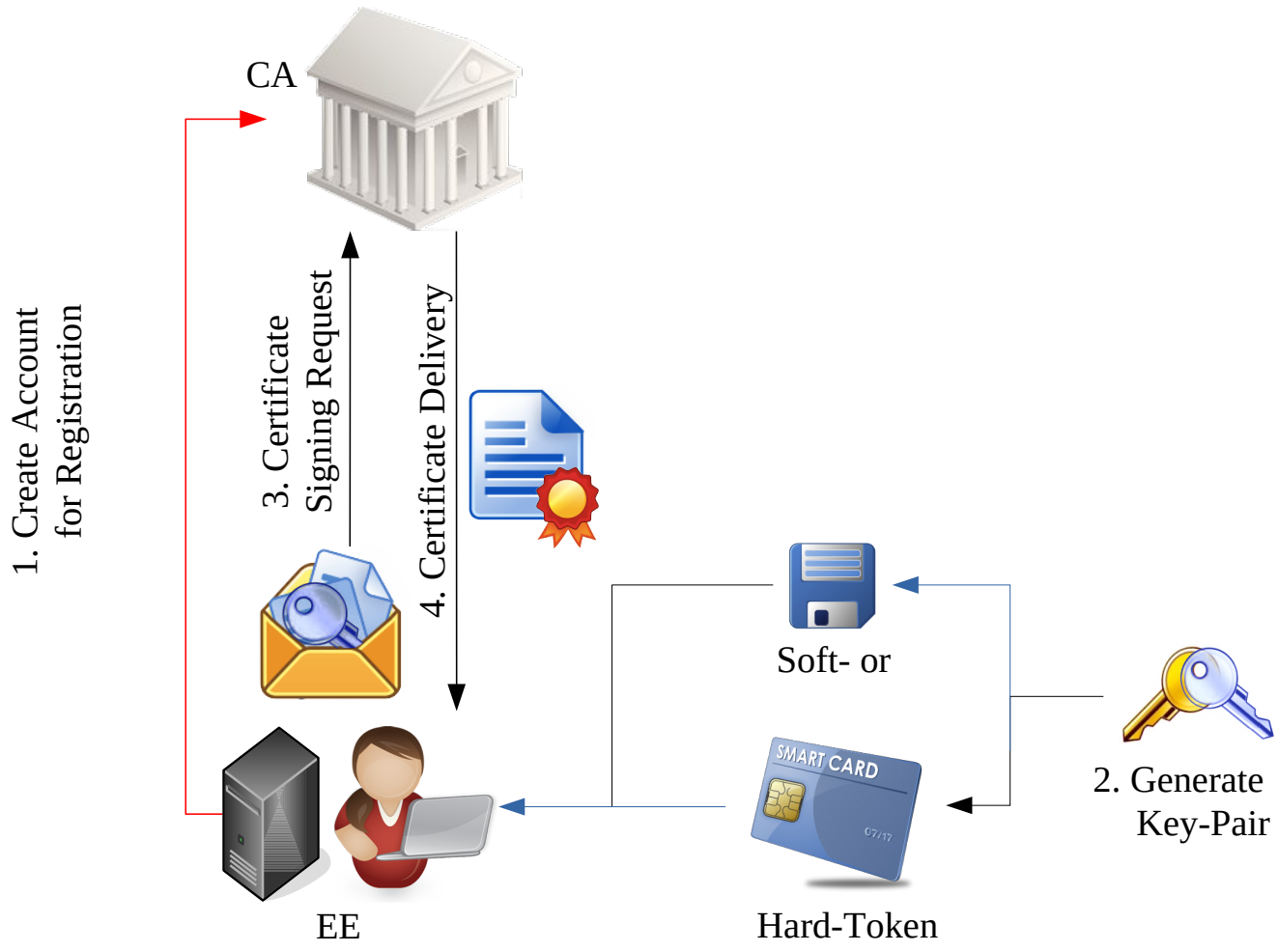
Legitimate Reasons for CA Key Backup/Recovery

- recovery of encrypted data after loss of access to private key (e.g. forgotten password, employee has left company, malfunctioning smart-card, corrupted disk (where private keys are stored), ...)
- processing of encrypted messages for policy enforcement (i.e. by anti-virus) at the network perimeter

Key Escrow

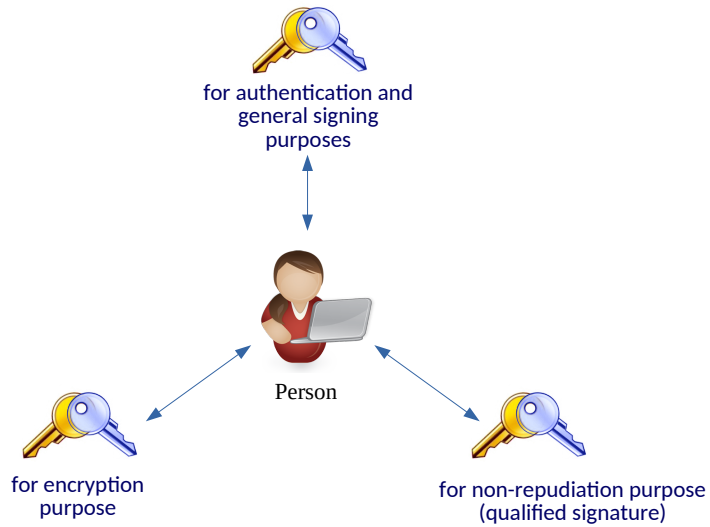
- Key escrow is a means to rebuild cryptographic keys in case the legitimate entity loses access to the data through a disaster or accident.
- The key does not have to be released to anyone other than the EE itself.
- However, key escrow may also be abused by a government for surveillance purposes ...

Key Generation by End-Entity (EE)



⇒ Key backup by EE only

Generating multiple key pairs per End-Entity



- In a private PKI environment a single multipurpose key pair might be suitable.
- Public PKIs normally provide multiple key pairs for different applications.
- Multiple key pairs are often used to support special policies for distinct services.

Generating multiple key pairs per End-Entity

Dual or multiple key pairs are issued to enable the usage for different purposes and to support special policies for various services.
Main reasons for multiple keys include:

- Different registration processes
- Restriction to certain policies and roles of EE
- Location of key generation and storage facility (e.g. must be generated on a secure signature creation device (SSCD))
- Security aspects (qualified signature $\leftrightarrow$ authentication) to prevent challenge semantic attacks
- Backup and recovery policies
- Key history and archiving handling

Procedures after Key Expiration

1. Expired private keys used **for encryption** need to be accessible even after the expiration or revocation of a certificate.
   $\rightarrow$ Long-term storage of expired **private encryption keys**!
2. Digitally signed documents have to be **verifiable** also after the expiration or revocation of a certificate. All necessary information used for the validation of a signature has to be present for a long term period.
   $\rightarrow$ Long-term storage of expired **public verification keys** and certificates.

Be aware: the use-case matters in both cases!

The Onboarding Process

- Crucial non-technical process in a PKI world.
- Dependent of the purpose and field of application of a certificate this process may be automatically / simple up to time consuming / complex.
- In a enterprise environment with a private PKI this is mostly done automatically by an Identity Management System (IdMS).
- For public PKI's the range is very broad:
  - Simple e-mail registration
  - Identification by sending a copy of identification card
  - Personal appearance up to a rigid vetting of the identity.

How to request a certificate from a PKI?

The main certificate request protocols are:

- The Public Key Cryptography Standard has defined a Certification Request Syntax Standard (PKCS#10) in combination with a secure channel (SSL/TLS) or the Cryptography Message Syntax (PKCS#7).
- The Simple Certificate Enrollment Protocol (SCEP) Originally specified by Cisco used for certificate auto-enrollment for network devices. Today used by Mobile Device Management solutions (MDM).
- Microsoft has specified a Auto-Enrollment process for Domain integrated Entities.
- Auto-Enrollment specified by Microsoft used in AD. The Certificate Request Message Format (CRMF) specified in RFC 4211 as part of the Certificate Management Protocol (CMP), often used in parallel with PKCS#10.

The Certificate Management Protocol (CMP) [2]

CMP is an Internet protocol specified to manage X.509 digital certificates within a PKI:

- A CMP client is able to communicate with a PKI Service to request, revoke, suspend and resume certificates.
- CMP messages are ASN.1/DER encoded and are usually encapsulated in HTTP(S) messages.
- CMP is supported by several libraries: cryptlib, EJBCA, OpenSSL, BouncyCastle, ...

Automatic Certificate Management Environment (ACME)

ACME allows setting up a HTTPS server and automatically obtain a browser-trusted certificate, without any human intervention.

- Runs a certificate management agent on the Web server → e.g. Certbot
- Issues Domain Validated (DV) certificates for Web servers only
- Certificates have a lifetime of 90 days

`https://letsencrypt.org/how-it-works/`

Downside of the Trust-list Model

- There are hundreds (!) of "trusted" Root CAs installed in the OS/Browser.
  - There are hundreds of browser-accepted Root CAs and an unknown number of subordinate ICAs
  - Each of them can break TLS/X.509 security
  - It does not matter how good your CA is — the only thing that matters is the worst CA of them all!
- The set of trusted Root CA's must be protected against malicious modification.
- The set of trusted Root CAs must be updated
  - automatically (MS, Apple, ...),
  - or on next release (Mozilla, others).
- Who decides which CA's are trusted?

We assume software vendors can establish trustworthy CAs!

Part II: Trust Issues in X.509

Certificate validation is hard

- BERserk was a catastrophic failure in the certificate validation of the NSS library (used by Firefox / Chrome)
- Most TLS libraries had a chain validation issue at some point

Certificate validation is hard

- BERserk was a catastrophic failure in the certificate validation of the NSS library (used by Firefox / Chrome)
- Most TLS libraries had a chain validation issue at some point

Let's assume TLS is correct and correctly implemented...

X.509 CA challenges

- Must trust a CA
  - Which one?
  - What is it trusted to do?
- Certificate bindings must be correct
  - Which John Smith is this?
  - Who authorizes attributes in a certificate?
  - How long are these values valid?
  - What process is used to verify the key holder?

Legal Aspects

For what is a CA liable?

- Certificate policies (CP) define rights, duties and obligations of each party in a PKI
- These documents usually have a legal effect
- The CP should be publicly exposed by CAs on their Web site and include:
  - Registration procedures
  - Revocation procedures
  - Liability issues

Are CAs trustworthy?

- Any public CA can issue certificates for any domain name without obtaining permissions.
- Who are these organizations, which we allow to issue certificates with little supervision?
- Most public CAs follow political and economic interests.

Furthermore, RCAs signed certificates of Intermediate CAs (ICA) with no constraints. A study (2013) in the wild revealed:

- 19 Root CAs signed $\approx$ 15'430 ICAs
- Only 94 ICAs ware owned by the signing CA organizations!

CA issues all the time

- June 2013: ANSSI issues certs for Google
- March 2014: India CCA intermediate compromised and issued certs for Yahoo and Google
- Feb 2015: Superfish / Privdog / Komodia breaking certificate authentication
- March 2015: Comodo cert for live.fi through hostmaster@live.fi
- March 2015: Same thing for xs4all
- March 2015: Google found bad certs issued by MCS Holdings / CNNICa
  - CNNIC issued intermediate certificate to Egyptian company MCS Holdings
  - MCS used it in a Man-in-the-Middle-TLS-Proxy in violation of policy
- April 2015: Google and Mozilla remove CNNIC

There where many more Root CA incidents in the past:

`http://wiki.cacert.org/Risk/History`

Domain Validation via E-Mail

- Domain Validation: CA sends mail to defined aliases (admin, administrator, webmaster, hostmaster, postmaster, see Baseline Requirements)
- If you offer E-Mail you **must** make sure that nobody can register such an address
- One can argue if this is sane system, but it is documented (Baseline Requirements)
- live.fi / xs4all.nl issues were their fault

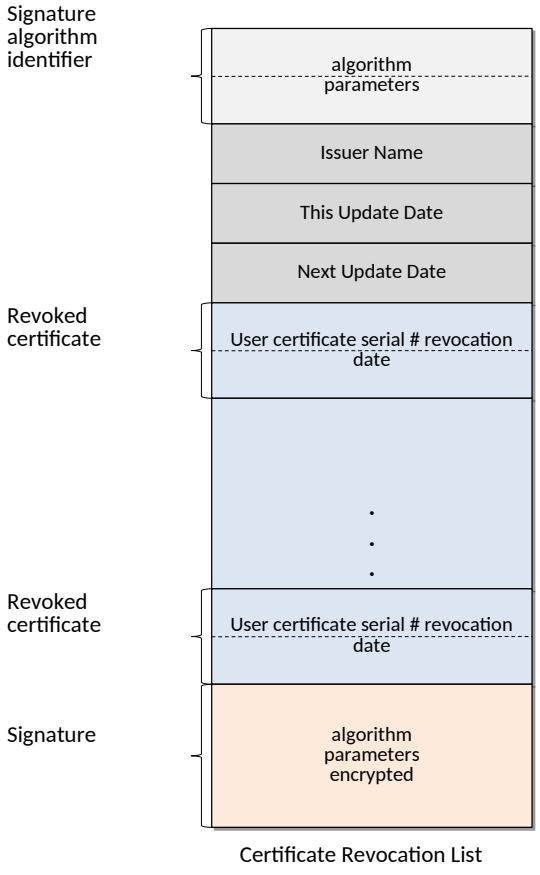
Certificate Revocation

- Sometimes it is necessary to immediately block usage of a certificate before its expiration date.
- A revocation request allows an EE (or issuing CA) to revoke a still valid certificate.
- The CA must publish the revocation subsequently on a specific infrastructure
- Two realization concepts:
  - *Black-lists* via Certificate Revocation List (CRL) [4], and
  - *White-lists* via Online Certificate Status Protocol (OCSP) [10]

Certificate revocation lists (CRLs) [4]

CA must maintain and publish a list of all revoked (but not expired) certificates. Revocation reasons include:

- Users private key was compromised (e.g. user lost private key)
- Content of certificate has changed (e.g. user has left enterprise)



Online Certificate Status Protocol (OCSP) [10]

Client-Server Architecture (uses port 80):

Clients: Check certificate status online by a OCSP Responder, can ask multiple certificates per query

Responder: Replies with a signed message using a certificate with extension:

$$\text{extendedKeyUsage} = \text{OCSPSigning}$$

Information per certificate:

- good
- revoked
- unknown (e.g. certificate outdated)

Signed responses may be stored/cached.

Drawbacks of OCSP

Performance and resource issues:

- OCSP may provide significant cost to a CA:
  - High traffic website may generate huge volume of OCSP requests.
  - OCSP slows down browsing, since it requires the client to contact a third party (the CA) to confirm the validity of each certificate that it encounters.
- Privacy:
  - OCSP checking potentially leaks user privacy information to third party online OCSP service.
- User (in)decision:
  - If OCSP response fails, user often is challenged by incomprehensible options ...

OCSP Stapling in TLS

OCSP stapling in TLS mitigates these problems:

- The certificate holder queries the OCSP server itself at regular intervals to obtain a signed time-stamped OCSP response.
- When browser of the client attempt to connect to the site, this response is included ("stapled") within the TLS handshake (ServerHello extension).
- The TLS client must explicitly request OCSP information as part of the ClientHello message

Revocation in practice

- Browsers use insecure soft-fail mode (in the past?)
- Chrome and Firefox distribute their own blocklists, but they don't scale
- OCSP stapling could help, but needs a mechanism to indicate its use (muststaple draft)

Revocation in practice

- Browsers use insecure soft-fail mode (in the past?)
- Chrome and Firefox distribute their own blocklists, but they don't scale
- OCSP stapling could help, but needs a mechanism to indicate its use (muststaple draft)

	Desktop Browsers											Mobile Browsers				
	Chrome 66			Firefox 60			Opera		Safari	IE	Edge	Safari	Chrome		Firefox	
	OS X	Lin.	Win.	OS X	Lin.	Win.	OS X	Win.	11	11	42	iOS	iOS	And.	iOS	And.
Request OCSP response	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Respect OCSP Must-Staple	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
Send own OCSP request	✗	✗	✗	-	-	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	-

(`https://blog.apnic.net/2019/01/15/is-the-web-ready-for-ocsp-must-staple/`, 2019)

Revocation in practice

- Browsers use insecure soft-fail mode (in the past?)
- Chrome and Firefox distribute their own blocklists, but they don't scale
- OCSP stapling could help, but needs a mechanism to indicate its use (muststaple draft)

	Desktop Browsers											Mobile Browsers				
	Chrome 66			Firefox 60			Opera		Safari	IE	Edge	Safari	Chrome		Firefox	
	OS X	Lin.	Win.	OS X	Lin.	Win.	OS X	Win.	11	11	42	iOS	iOS	And.	iOS	And.
Request OCSP response	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Respect OCSP Must-Staple	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
Send own OCSP request	✗	✗	✗	-	-	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	-

(https://blog.apnic.net/2019/01/15/is-the-web-ready-for-ocsp-must-staple/, 2019)

https://scotthelme.co.uk/lets-encrypt-to-end-ocsp-support-in-2025/

Man in the Middle Proxies

- Superfish: Created a TLS Man in the Middle Proxy, private key was static and part of the Software (Komodia)
- Privdog: Just disabled TLS verification completely (Privdog is founded by the CEO of Comodo)
- Several Antiviruses do the same. Not fully broken, but all decrease the security of TLS
- This is not directly a problem of CAs or TLS

Part III: X.509 CA Alternatives

How to identify black sheeps?

Because its hard to prevent the issuance of forged certificates...

... we must give power to the public to detect malicious EEs with faked certificates.

$\Rightarrow$ For browsers able to verify certificates, several alternative or additional ways have been proposed.

Alternative 1: DNSSEC/DANE

DNS-based Authentication of Named Entities (DANE).

`https://tools.ietf.org/html/rfc6698`

Alternative 1: DNSSEC/DANE

DNS-based Authentication of Named Entities (DANE).

`https://tools.ietf.org/html/rfc6698`

DANE will not provide you any security today.

It is very uncertain if it will ever do that.

DNSSEC — too many pieces

For DNSSEC to work you need:

- A signed root
- A signed Top Level Domain
- A domain broker that supports DNSSEC
- A DNS operator that supports DNSSEC
- A client that verifies DNSSEC

Only if you have all five you have security.

Working DNSSEC deployment is near zero

- DNSSEC propaganda: "xx % of all TLDs are signed", "there are already XX.XXX signed domains"
- Completely irrelevant statements
- Cryptographic signatures are not worth anything if nobody is checking them
- Once you enable checking, you find out signatures are invalid
- Even if they are valid today, that may not be true after key rollover
- Same issue as with TLS: How many users are you willing to burn?

$\Rightarrow$ Client deployment of DNSSEC is very close to zero

DNSSEC client

- So how exactly does a client verify DNSSEC signatures? (Most common today: Not at all)
- DNSSEC verification happens in the DNS resolver — but clients usually do not have full DNS resolvers

DNSSEC client

- Should we trust our providers? (No!)
- Should operating systems ship DNS resolvers?
- Should applications ship their own DNS resolvers?
- Not clear how DNSSEC should be deployed on clients!

So what is DANE?

- Idea of DANE: If we already have a secure DNS through DNSSEC we can add certificate information to the DNS
- The problem: We do not have working DNSSEC
- Building something on top of something that does not work is pointless
- Also, to secure DNS, IETF proposed putting DNS-over-TLS

Does anyone see a chicken-and-egg problem here?

Alternative 2: HTTP Public Key Pinning (HPKP)

- Webpage sends a header with hashes of public keys for the browser to pin
- Browser stores these hashes
- Always needs at least two keys - because you need to be able to change your certificates in the future
- Adds a "Trust on First Use" (ToFU) protection

`https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning`

HTTP Public Key Pinning (HPKP)

HPKP header:

```
Public-Key-Pins: max-age=31536000;
pin-sha256="HD3EpAqgxJWKGiSuuXPyipmL33IwYlwhLUgF1gKYOuc=";
pin-sha256="dwUkkREEnv6pEtNJoRzlBHJm3IlUvPhgy0mdYFOM6V8=";
includeSubDomains; report-uri="/hpkp.php"
```

- Browser pins the two hashes for [max-age] seconds

HPKP: Only for HTTPS

- One big drawback: It was only for the Web
- Needed for deployment: Software change in browsers and configuration change on servers
- As HPKP is implemented via HTTP headers it does not work on other protocols
- There is a proposal called TACK to do something similar on the TLS layer

HPKP Warning

HPKP improves confidentiality, but can be dangerous to availability:

- If you loose your keys you may lock out your visitors!
- Needs careful planning of key management.
- HPKP “suicide” attack

HPKP deployment

- HPKP **was** supported by Chrome/Chromium and Firefox

`https://blog.qualys.com/ssllabs/2016/09/06/is-http-public-key-pinning-dead`

HPKP deployment

- HPKP **was** supported by Chrome/Chromium and Firefox

`https://blog.qualys.com/ssllabs/2016/09/06/is-http-public-key-pinning-dead`

`https://scotthelme.co.uk/hpkp-is-no-more/`

Alternative 3: Certificate Transparency [9]

- Public logs with all certs in them
- Certificate can contain log proof confirming that it has been added to a log
- When a browser sees a certificate that is not in the log it can raise alarm
- Certificate Transparency runs in soft-fail mode, it cannot prevent misuse
- But it makes it hard to use malicious certificates without being noticed

`https://www.certificate-transparency.org`

Alternative 4: Certification Authority Authorization (CAA)

- CAA tells the client which CA is allowed for a domain
- Relies on DNS(SEC) security
- The configured CA must also still be trustworthy

Alternative 5: HTTP Strict Transport Security (HSTS)

- HSTS tells the browser to mark a page as HTTPS only for a defined timeframe
- Further prevents stripping attacks
- You can even pre-load your webpage as HTTPS only into Chrome and Firefox

HSTS attack through NTP

- HSTS protects a page for a defined timeframe
- System time is considered trustworthy, but it is not!
- Delorean-Attack circumvents HSTS with NTP
- NTP provides no security (solutions: tlsdate, openntpd)

Part IV: Digital Timestamping

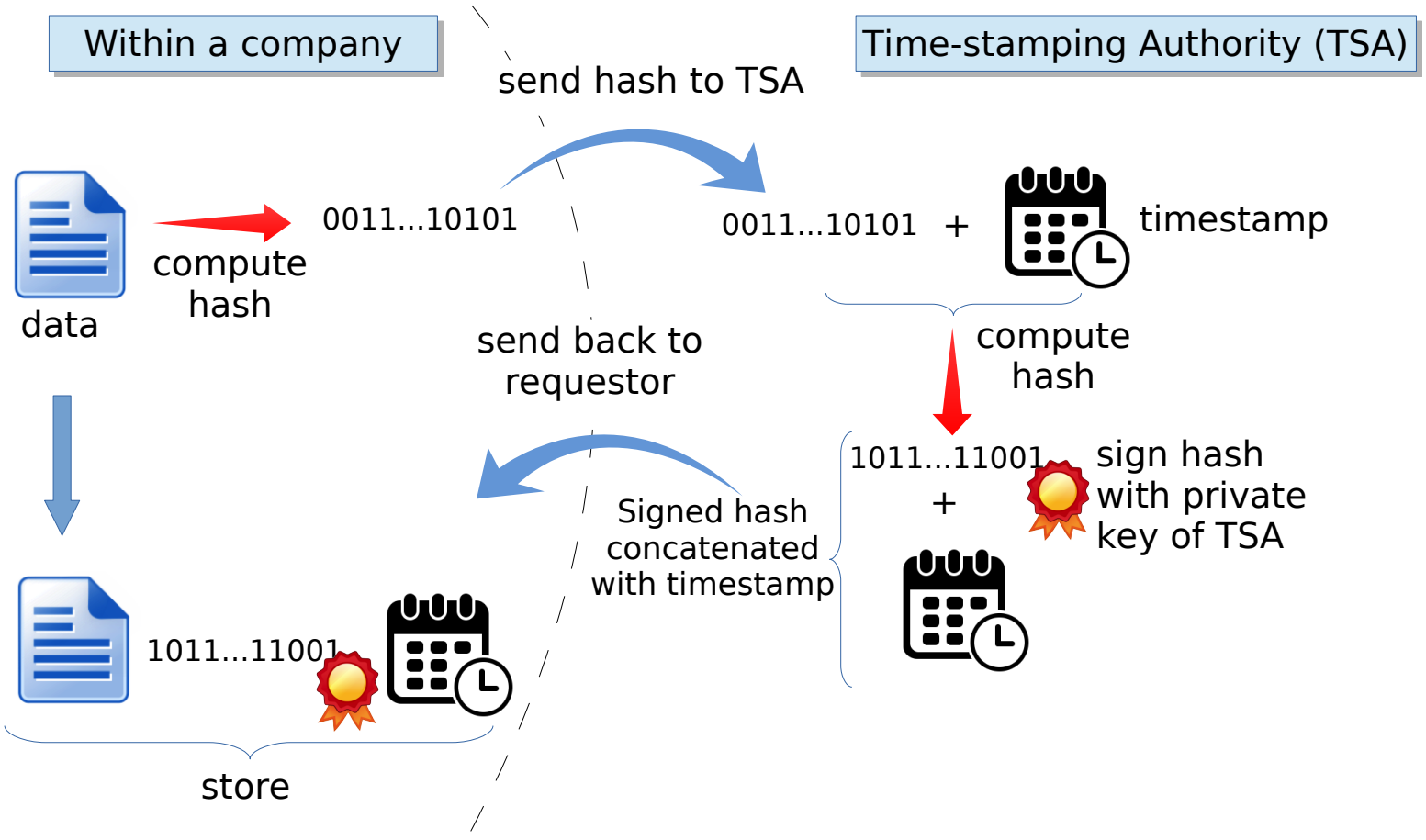
Motivation

A "trustworthy time" is crucial for a PKI.

- CAs should issue/revoke certificates based on a trustworthy time. This service is mandatory for accredited CAs issuing advanced or qualified certificates.
- A trustworthy timestamp should be added every time a user "digitally signs" a document to achieve a higher level of legal probative value.

If the content of a document is additionally digitally time-stamped by a trustworthy DTS, it makes time of creation provable in the future.

Timestamp Protocol (TSP) [1]



Part V: Trust Agility

Guiding questions "SSL and the Future of Authenticity"

- What is fundamentally wrong with the current CA model?
- What is the idea of "trust agility", and is it reasonable?
- Understand the notion of "perspectives". Evaluate strengths and weaknesses of the perspective model.

Interlude: SSL and the Future of Authenticity

References I

C. Adams, P. Cain, D. Pinkas, and R. Zuccherato.
Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP).
RFC 3161 (Proposed Standard), August 2001.
Updated by RFC 5816.

C. Adams, S. Farrell, T. Kause, and T. Mononen.
Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP).
RFC 4210 (Proposed Standard), September 2005.
Updated by RFC 6712.

S. Chokhani, W. Ford, R. Sabett, C. Merrill, and S. Wu.
Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework.
RFC 3647 (Informational), November 2003.

References II

D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk.
Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.
RFC 5280 (Proposed Standard), May 2008.
Updated by RFC 6818.

C. Evans, C. Palmer, and R. Sleevi.
Public Key Pinning Extension for HTTP.
RFC 7469 (Proposed Standard), April 2015.

P. Hallam-Baker and R. Stradling.
DNS Certification Authority Authorization (CAA) Resource Record.
RFC 6844 (Proposed Standard), January 2013.

References III

J. Hodges, C. Jackson, and A. Barth.
HTTP Strict Transport Security (HSTS).
RFC 6797 (Proposed Standard), November 2012.

P. Hoffman and J. Schlyter.
The DNS-Based Authentication of Named Entities (DANE)
Transport Layer Security (TLS) Protocol: TLSA.
RFC 6698 (Proposed Standard), August 2012.
Updated by RFCs 7218, 7671.

B. Laurie, A. Langley, and E. Kasper.
Certificate Transparency.
RFC 6962 (Experimental), June 2013.

References IV

S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams.
X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP.
RFC 6960 (Proposed Standard), June 2013.

J. Schaad.
Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF).
RFC 4211 (Proposed Standard), September 2005.

Further reading I

- Google on CNNIC http://googleonlinesecurity.blogspot.com/2015/03/maintaining-digital-certificate-security.html
- Mozilla on CNNIC https://blog.mozilla.org/security/2015/04/02/distrusting-new-cnnic-certificates/
- live.fi bad cert https://technet.microsoft.com/en-us/library/security/3046310
- xs4all bad cert https://raymii.org/s/blog/How_I_got_a_valid_SSL_certificate_for_my_ISPs_main_website.html
- OCSP muststaple https://tools.ietf.org/html/draft-hallambaker-muststaple-00
- Superfish https://noncombatant.org/2015/02/21/superfish-round-up/

Further reading II

- Privdog https://blog.hboeck.de/archives/865-Software-Privdog-worse-than-Superfish.html
- Why not DNS records (Ryan Sleevi) https://lists.w3.org/Archives/Public/public-webappsec/2014Dec/0264.html
- DNSSEC is dead (Alex Stamos) http://www.slideshare.net/astamos/appsec-is-eating-security
- Against DNSSEC (Thomas Ptacek) http://sockpuppet.org/blog/2015/01/15/against-dnssec/
- SSL Strip http://www.thoughtcrime.org/software/sslstrip/
- HSTS Preload https://hstspreload.appspot.com/
- Bypassing HTTP Strict Transport Security https://www.blackhat.com/docs/eu-14/materials/eu-14-Selvi-Bypassing-HTTP-Strict-Transport-Security-w pdf

Further reading III

- Delorean NTP MitM
  `https://github.com/PentesterES/Delorean`