Establishing a Certification Authority (CA)

Version 1.1

Publication

Document Revision History

Date	Version	Description
February 2023	1.0	Initial Release
September 2023	1.1	• Minor corrections • Added further certificate validation types in chapter 2.1 • Added the new CA/B Forum Baseline Requirements for S/MIME certificates

This handbook has been developed to provide an overview of how to establish a Certification Authority. The content is based on public information and not solely the view of the NRCA and ETDA. It is a guideline and may be updated from time to time.

Third party sources are quoted as appropriate. ETDA is not responsible for the content of the external sources, including external websites, nor their continued availability, referenced in this handbook.

Where specific vendors or product names are given, those do not mean endorsement from ETDA, but serve as examples only.

This handbook is intended for educational and information purposes only. Neither ETDA nor any person acting on its behalf is responsible for the use that might be made of the information contained in this handbook. All information contained herein is provided on an "As Is" basis with no warranty whatsoever. NRCA/ETDA does not promise any specific result, effects, or outcome from the use of the information herein.

Written by Martijn van der Heide

[1] Creative Commons License: <https://creativecommons.org/licenses/by-nc-sa/4.0/>

Table of Contents

Preface

Communications, in terms of both the messages themselves and their transport, are increasingly conducted in electronic form and use the Internet for easy, fast, cheap, and global delivery. Such electronic interactions these days include much more than just e-mails and bulletin board postings: indeed, every aspect of our day-to-day lives have moved online, either to complement our regular offline activities (such as talking to our family and friends), or in its entirety (working from home, online gaming or ordering from shops).

All electronic communications require a certain amount of security and trust. While we have, thankfully, mostly left the era behind where infrastructure shortcomings caused significant problems to get data across without requiring several retries (do you remember modems?), there are still several things that need our attention. Who can see your messages in transit? How can you safely authenticate a user coming to your web portal? How can you make sure that the message was not altered before (or even after) arriving? How can you prove that it was you who sent the message?

Cryptography is the science of writing or reading coded messages; it is the basic building block that enables the mechanism of authentication (which establishes the identity of the sender or receiver of information, or both), integrity (which ensures that the data has not been altered in transit for at rest) and confidentiality (which ensures that only authorized entities can view the data). Digital certificates are used to attest the validity of a public key that's part of asymmetric encryption and a common format is the X.509 standard.

Security certificates themselves are just small files, composed of a block of unique data generated by a cryptographic algorithm which is then bound to an identity (organization, person, or asset). Because anyone can create certificates with the right software, a second component is needed to create trust: public signing of the certificate by a trusted party. We call this party a CA: Certification Authority, or Certificate Authority (both names are used interchangeably). A CA is also responsible for the revocation of certificates if they cannot be trusted anymore for any reason. The worldwide implementation of such certificate use is called Public Key Infrastructure, or PKI.

This handbook looks at what a Certification Authority is, how its role fits into the larger PKI ecosystem, and how an organization could set one up, either as a Private (Internal) or a Public Root CA.

Intended audience

This handbook is designed for anyone who wishes to learn more about Certification Authorities (CA) and perhaps start one themselves. It describes both the process to establish a CA and the various requirements and considerations to be made. A worked out example is also provided, to show how some of the various components fit together.

The intended audience is senior security management and architects, but the handbook can also be used as a reference guide for operational staff.

Structure of this handbook

Chapter 1 introduces PKI. After a brief introduction to the relevant cryptographic concepts, it describes what the various components and parties of the PKI ecosystem are, what a CA hierarchy might look like and how trust is anchored in the Chain of Trust or the Web of Trust.

Chapter 2 is the heart of this handbook and shows the steps and requirements to set up any CA, either a Private (Internal) or a Public Root CA. It describes the various types of common digital certificates, the Certificate Policy (CP) and Certification Practice Statement (CPS) documents, infrastructure and other security considerations, Certificate Management Systems, Certificate Transparency, and looks at how to deal with serious problems. It also introduces the industry: the Common CA Database (CCADB) and CA/Browser Forum.

Chapter 3 discusses further requirements and steps, on top of chapter 2, for Public Root CAs. This includes annual WebTrust audits, quality assurance, regulation and, perhaps most importantly: applying to the Trust Stores to get your root certificate trusted in applications such as web browsers worldwide.

Chapter 4 is a worked out example of a Private CA built with OpenSSL. If follows the concepts as set forth in the previous chapters, allowing the reader to see in detail how it all fits together.

Chapters 5 and 6 move into the world of regulatory requirements in the European Union. Chapter 5 introduces the eIDAS regulation and what the security requirements are to make sure all CAs in the EU conform to the same high standard and are fully interoperable, while chapter 6 introduces a recent addition to that, where all citizens in the EU will have a personal e-Identity.

Several appendixes have been added to provide further references to the material discussed in this handbook.

Acknowledgements

Special thanks go to all individuals in the international information security community who kindly peer reviewed this handbook.

Acronyms

This handbook uses the following acronyms:

Acronym	Term
ACME	Automatic Certificate Management Environment
AEG	Auto Enrollment Gateway
AES	Advanced Encryption Standard
CA	Certification Authority, or Certificate Authority
CAA	Certification Authority Authorization
CCADB	Common CA Database
CEP	Certificate Enrollment Protocol
CP	Certificate Policy
CPS	Certification Practice Statement
CRL	Certificate Revocation List
CSR	Certificate Signing Request
CT	Certificate Transparency
DES	Data Encryption Standard
DNS	Domain Name System
eIDAS	(European Union) electronic IDentification, Authentication, and trust Services
ETSI	European Telecommunications Standards Institute
FIPS	(US Government) Federal Information Processing Standards
FQDN	Fully Qualified Domain Name
HSM	Hardware Security Module
IoT	Internet of Things
LDAP	Lightweight Directory Access Protocol
MD4/5/6	Message-Digest Algorithm
MMD	Maximum Merge Delay
OCSP	Online Certificate Status Protocol
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
QTS	Qualified Trust Service
QTSP	Qualified Trust Service Provider
RA	Registration Authority
RFC	Request For Comments
RSA	Rivest–Shamir–Adleman
SCT	Signed Certificate Timestamp
SCEP	Simple Certificate Enrollment Protocol
SHA-1	Secure Hash Algorithm
SSL	Secure Sockets Layer
TLS	Transport Layer Security

TPM	Trusted Platform Module
TSP	Trust Service Provider
V2X API	Vehicle-to-everything
VA	Validation Authority
WinEP	Windows Enrollment Proxy

Table 1: List of acronyms

1 Introduction to PKI

Public Key Infrastructure is a complete framework rather than just a protocol. The term consists of 2 parts: Public Key and Key Infrastructure.

A public key infrastructure (PKI) is a set of roles, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption.

1.1 Relevant cryptographic concepts

Before discussing PKI, let us introduce several concepts first. Note that, as technology moves forward, current accepted encryption algorithm standards will become unsafe – with the ever increasing computer power, the algorithms can be defeated over time. That is why new algorithms continue to evolve. The speed of this problem will increase exponentially when quantum computing becomes available.

1.1.1 Symmetric-key encryption

Symmetric-key encryption is a type of encryption where only one key (a secret key) is used to both encrypt and decrypt electronic data. The entities communicating via symmetric encryption must exchange the key so that it can be used in the encryption and decryption process, hence why this key is also called a shared secret.

By using symmetric-key encryption algorithms, data is "scrambled" into what is called ciphertext, so that it cannot be understood by anyone who does not possess the secret key to decrypt it. Once the intended recipient who possesses the key has the message, the algorithm reverses its action so that the message is returned to its original readable form.

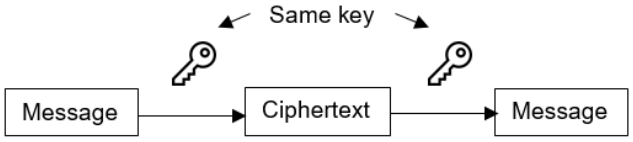

Figure 1: Symmetric-key encryption

Symmetric-key encryption is most often used for data confidentiality because most symmetric-key algorithms have been designed to be implemented in hardware and have been optimized for encrypting large amounts of data at one time. Challenges with symmetric-key encryption include:

1) changing the secret keys frequently to avoid the risk of key compromise;
2) securely generating the secret keys;
3) securely distributing the secret keys.

Symmetric-key encryption algorithms are usually better for bulk encryption. They have a smaller key size, which means less storage space and faster transmission. Due to this, asymmetric-key encryption is often used to exchange a secret session key for symmetric-key encryption such as in SSL/TLS.

Examples of symmetric-key encryption are AES, DES and 3DES, but DES is considered unsafe to use.

1.1.2 Asymmetric-key encryption

In asymmetric-key encryption, there is a pair of keys for one identity:

- a private key.
- a public key.

The private key must not be shared to others, while the public key can be shared with anyone.

When you encrypt a message with the public key of the recipient, then only the recipient can decrypt with their private key assuring confidentiality.

Since public keys are widely available, anyone can pretend to be the sender. If the sender first encrypts with the recipient's public key and then encrypts with their own private key, the recipient can be assured that the message originated from the sender.

Public key algorithms are rarely used for data confidentiality because of their performance constraints. Instead, public key algorithms are typically used in applications involving authentication using digital signatures and key management.

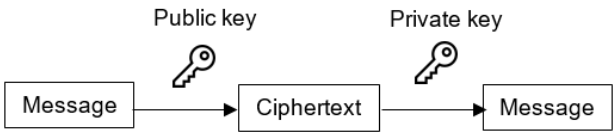

Figure 2: Asymmetric-key encryption

Examples of asymmetric-key encryption are ElGamal, RSA and Elliptic curve.

1.1.3 Cryptographic hash function

A hash function is a one-way cryptographic function that takes an input message of arbitrary length and outputs a fixed length code. The output is called a hash or message digest.

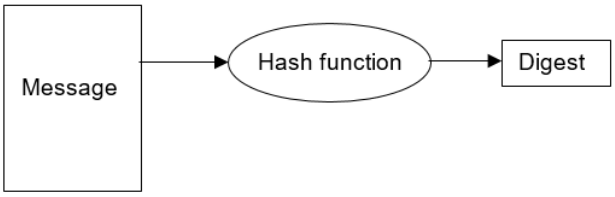

Figure 3: Hash function

A strong cryptographic hash algorithm will make it difficult to:

1) Guess the original input message that yielded the hash (pre-image resistance).
2) Find two or more input messages that yield the same hash (collision resistance).

Examples of cryptographic hash functions are MD4/5/6, SHA-1 and SHA-512, but MD4/5 and SHA-1 are considered unsafe to use.

1.1.4 Digital signature

A digital signature serves a similar purpose as a real world signature. Its purpose is twofold: ensure the signer's identity and prove that the message has not been altered in any way after it left the hands of the signer.

The digital signature is generated in 2 steps:

1) Create a digest of the original message.
2) Encrypt that digest with the private key of the sender.

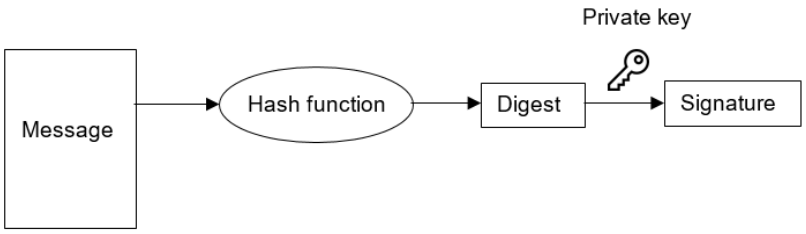

Figure 4: Digitally signing a message

This signature is then delivered together with the original message, allowing verification at the recipient's end.

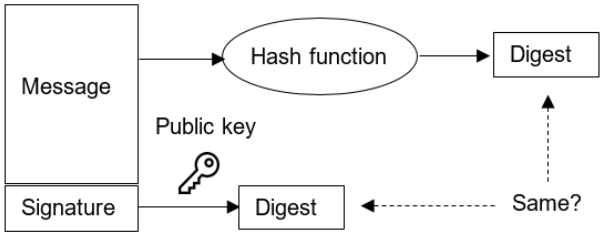

Figure 5: Verifying a digital signature

1.1.5 Digital certificate

A digital certificate, also known as a public key certificate, is used to cryptographically link ownership of a public key with the entity that owns it. Digital certificates are for sharing public keys to be used for encryption and authentication.

A certificate has been digitally signed by a Certification Authority (CA). The information normally found in a certificate conforms to the X.509 v3 standard as defined in RFC 8250[2].

Certificates conforming to the standard include information about the published identity of the owner of the corresponding private key, the key length, the algorithm used, and associated hashing algorithm, dates of validity of the certificate and the actions the key can be used for.

An individual may hold any number of certificates issued by any number of CAs.

[2] RFC 5280: https://www.rfc-editor.org/rfc/rfc5280

1.2 The components of PKI

There are several components involved in PKI, combining services and technologies. All those roles can be performed by the same organization, but some of them may be outsourced (with limitations).

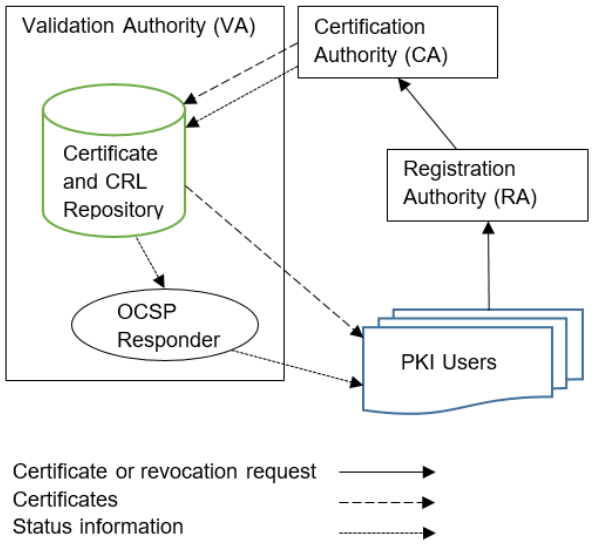

Figure 6: Relationship of the various PKI components

1.2.1 Certification Authority (CA)

The Certification Authority, or Certificate Authority (both names are used interchangeably), is the party that signs digital certificates.

There are 2 types of CAs: public and private. Public CAs are those vendors you can find on the Internet when you are looking to buy a certificate. Private (a.k.a. Internal) CAs are an IT asset located inside organizations, to create certificates needed by staff and devices for secure operations.

1.2.2 Registration Authority (RA)

The Registration Authority is the point of contact for anyone who wants to get their certificate signed or revoked by the CA. Signing requires a co-called Certificate Signing Request (CSR). The RA is responsible for verifying the request, the authenticity of the requester, ownership of the object for which a certificate is needed, and performing all other checks to make sure that the request is legitimate and complete.

The actual creation of the CSR file itself depends on your policy; either the Subscriber creates one themselves or the RA creates one on their behalf, or a combination thereof.

If the request is accepted, it will be passed to the CA to sign the certificate.

1.2.3 Validation Authority (VA)

Once a certificate has been signed, there may a reason to revoke the trust later, before it has expired, for example if the private key has been lost or compromised.

Such certificates are put on a Certificate Revocation List (CRL), made publicly available by the VA via either the CRL or OCSP protocol. All PKI-aware applications (e.g. web browsers) will automatically check the VA of the associated CA, to verify if a presented certificate has not been revoked.

The Online Certificate Status Protocol (OCSP) was created as a light-weight alternative to CRLs. Like CRLs, OCSP enables a requesting party (e.g., a web browser) to determine the revocation state of a certificate.

Certificate validation is a critical function in the Chain of Trust and should be implemented as highly-available, using a reverse proxy perhaps load balancers.

1.2.4 Auto Enrollment Gateway (AEG)

Certificate endpoints are no longer restricted to a specific device type. Today, the endpoints receiving certificates are diverse and distinctive, including IoT devices, telecom equipment, enterprise endpoints, applications, virtual functions, digital identities, and many more for a wide range of use cases involving digital identification, encryption, and non-repudiation.

With a diverse range of devices, the greatest issue for a CA is to distribute and manage the lifecycle of certificates. If the process is manual, it becomes impractical for huge organizations to handle at scale. The best illustration is IoT devices, which include millions of distinct devices dispersed over a large geographical area.

The optimal method for managing such an ecosystem is to automate the certificate life cycle management utilizing HTTP/s-based standard protocols. ACME, SCEP, CMP, V2X API, WinEP, etc., are the most prevalent and widely-used protocols.

There should be a gateway that acts as a front-end tier to interface with all devices using their supported standard enrollment protocols and to interact with the issuing CA for certificate issue, renewal, and validation.

Auto-enrollment and lifecycle management of the certificates result in a significant reduction in cost and administration simplicity with no human errors.

1.2.5 Certificate publishing methods

One of the fundamentals of PKI is the need to publish certificates so that users can find them (one must be able to get hold of the public key for the recipient of encrypted information). There are two ways of achieving this. One is for the CA to publish certificates in an X.500/LDAP-compliant directory. The other is to manually send your certificate out to those people.

1.2.6 Certificate Management System

At the heart of a CA, VA, RA, and AEG is a Certificate Management System. This platform is used for maintaining all certificates, to create, suspend, renew, revoke, and deploy certificates, to publish them as well as to populate the Certificate Revocation List.

1.2.7 Subscribers

A Subscriber is a person, legal entity, or infrastructure component whose name appears as the subject in a certificate. The Subscriber asserts the use of the key and certificate in accordance with the Certificate Policy asserted in the certificate.

Subscribers are also called End Entities, and their certificates are leaf certificates, which means they are at the end of the Chain of Trust.

1.2.8 Relying Parties

A Relying Party is an entity that relies on the validity of the binding of the Subscriber's name to a public key to verify or establish the identity and status of an individual, role, or system or device; the integrity of a digitally signed message; the identity of the creator of a message; or confidential communications with the Subscriber.

1.2.8.1 Trust Stores

Web browsers and Operating System vendors each independently maintain a Trust Store, also called Root Store or Trust List, of all Root CAs they decided to trust, after thorough inspection.

In addition, both Operating System and applications allow manually adding certificates that are trusted.

This will be explained in more detail in chapters 2 and 3.

1.2.8.2 PKI-aware applications

There is a growing amount of PKI-aware applications. Well-known are of course web servers, web browsers and e-mail clients, but the list includes many types of applications, such as PDF tools (document signing), app stores (code signing), DevOps tooling (code signing), and IoT devices.

1.3 Certification Authority hierarchy

Because the root certificate is the crown jewel of a Root CA, it needs a layer of protection from severe trust problems that may occur later: there must be at least 1 Intermediate CA between the Root certificate and Subscriber certificates. This allows the Root CA to simply revoke the Intermediate certificate instead of giving up the Root certificate altogether and ceasing operation.

These Intermediate CAs, or Subordinate CAs, operate under the trust of the Root CA and can potentially be operated by third parties. This can be used for segmentation; if there are several different use cases or services (certificate types) with different quality levels offered, each can be managed under a different Intermediate CA.

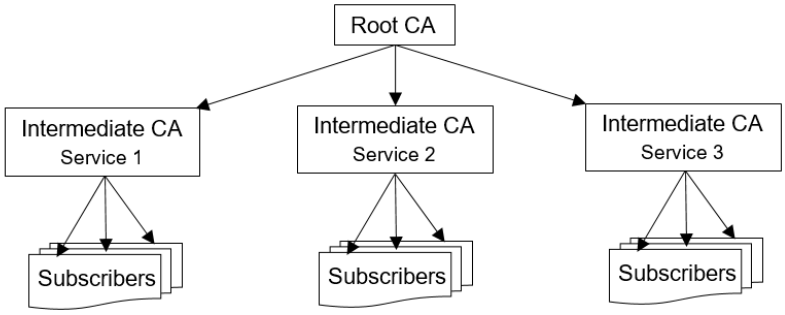

Figure 7: 2-Tier CA hierarchy

Test and production CAs should be strictly separated too, to ensure no test certificate accidentally leaks outside.

This concept can be extended with additional levels of Intermediate CAs, e.g., to also separate geographic regions.

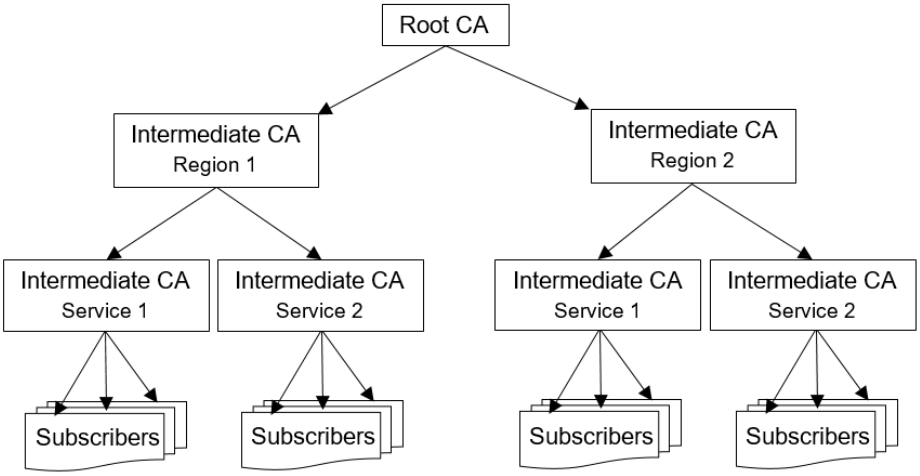

Figure 8: 3-Tier CA hierarchy

It is common for Public Root CAs to maintain several Roots, for different services or service levels.

1.4 Chain of Trust

Who is that CA who can sign and revoke certificates, and who decided to give them that trust? In other words, who signed the certificate from that trusted party, allowing them to further sign Subscriber certificates with it? And so on. This trust path, called the Chain of Trust, can be followed all the way back to a Root CA, who created the first certificate in that chain.

The root certificate should be available in the trust store, which is a collection of certificates that are trusted, and is often part of an operating system or web browser configuration and is installed locally on the client device.

Trust in a Subscriber certificate is verified by checking the certification path through all Intermediate certificates, back to a trusted Root certificate, installed in a trust store. This process is made easy by the fact that each certificate registered who the issuer was (its parent). PKI-aware software will also verify each certificate in the chain with the Validation Authority, to make sure it has not been revoked.

The steps to be followed to verify the Chain of Trust are the same for each level (Subscriber – Intermediate(s) – Root), as also shown in figure 9:

1) Verify the certificate's integrity, by verifying the signature with the public key of the issuer.
2) Verify the validity of the certificate, with the "notBefore" and "notAfter" fields.
3) Check the revocation status of the certificate.
4) Verify the issuer's name, which must match the subject name of the issuer's own certificate.
5) Verify the constraints specified in any certificate's extensions, such as domain name, policy, the key usage (purpose) of the certificate and the path length (to make sure leaf certificates are not illegally used as Intermediate certificates).

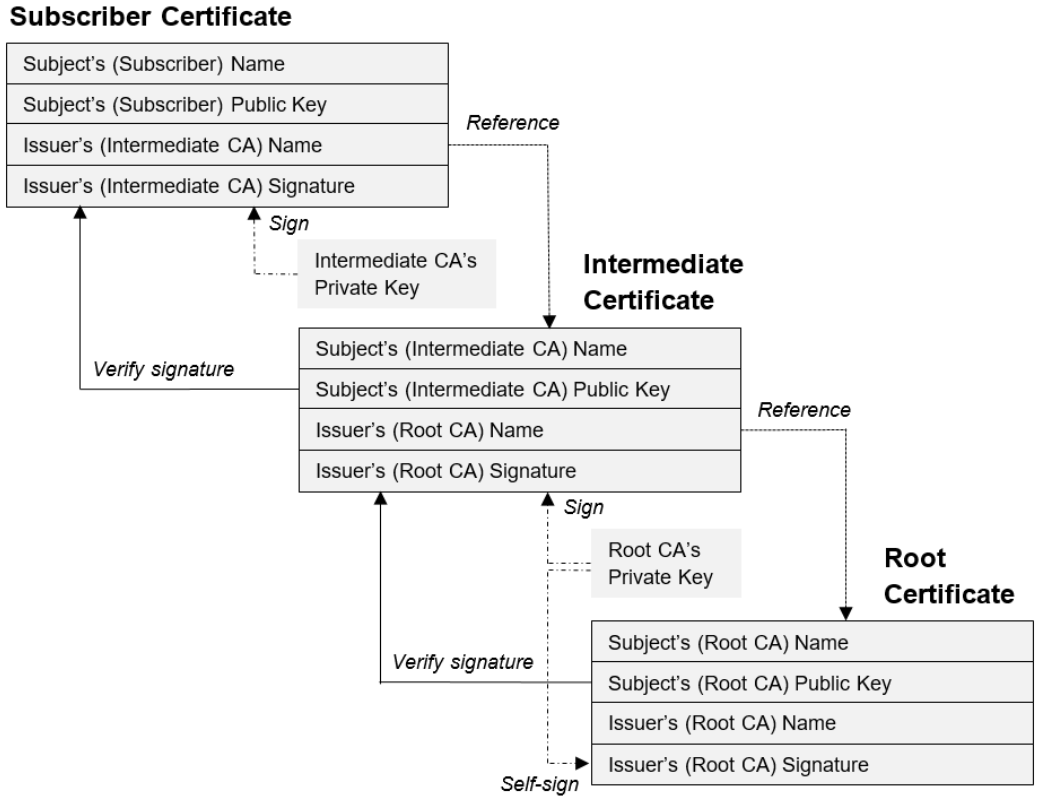

Figure 9: Chain of Trust

As you can see in the figure, Root certificates are always self-signed.

1.5 PGP and Web of Trust

PGP is a special type of PKI. It was the first practical implementation of PKI for public use, but has a slightly different philosophy.

Where, as explained above, PKI based on X.509 relies on centralized CAs, together with their RAs, to do all verification before signing, PGP does not have any central authority. Anyone can create a PGP key and upload it to one of the central Key servers. Availability of a key on a Key server does not mean that authenticity has been checked.

Since there is no Chain of Trust, PGP uses the so-called Web of Trust: other PGP users can do the verification of the key and its key holder, with government-issued photo-ID, and sign that key if verification passed successfully. This process is often performed at scale at side-events of gatherings of the security community (typically conferences and trainings), which are call Key Signing Parties.

After signing, the key should be uploaded to the Key server again, which activity adds the new signature to the key. Everyone can either obtain that key and check signatures in their PGP software, or see all signatures on the Key server.

2 Creating a CA

If you are setting up a Private (Internal) CA, then this chapter describes all steps.

If you are setting up a Public Root CA, then start with this chapter, and continue with chapter 3.

Creating a CA looks like a relatively straightforward project (but is a lot harder to accomplish than one might think) and involves the following steps

1) Decide which type(s) of certificate are to be issued
2) Create the Certificate Policy and Certificate Practice Statement
3) Build the infrastructure
4) Setup the Certificate Management System
5) Create the Root and Intermediate certificates
6) Distribute the Root certificate

Before starting, the use case you are trying to solve should be clear, otherwise setting up a full CA may be very expensive overkill. If your only goal is to issue SSL/TLS certificates for your internal network, it may be enough to set up a simple OpenSSL CA. This will be explored in chapter 4.

2.1 Certificate types

There are several types of Subscriber certificates, each with a different use case and/or trust level.

The most common certificate types are:

1) SSL/TLS server certificates

   An SSL/TLS certificate allows systems to verify the identity & subsequently establish an encrypted network connection to another system using the Secure Sockets Layer/Transport Layer Security (SSL/TLS) protocol. A typical use of such certificates is to provide HTTPS on web servers.

   1.1 Domain Validated (DV) certificates

   These are the standard certificates, where the customer only needs to demonstrate control of the domain or URL.

   1.2 Organization Validated (OV) certificates

   To obtain such a certificate, apart from the customer to prove control of the domain or URL, additional steps are taken to validate legitimacy of the organization itself.

   1.3 Extended Validated (EV) certificates

   EV certificates can be issued only by a subset of Certification Authorities (CAs) and require verification of the requesting entity's legal identity before certificate issuance.

1.4 Individual Validated (IV) certificates

For this type of certificate, the identity of the individual person listed as the Subject of the certificate. This may be the same as the person who requested the certificate. Often, the address of the Individual is also verified.

2) Client Authentication certificates

A client authentication certificate (Individual Validated, IV) is used by a person/device to authenticate their identity to a remote server while making an online request. The public key must be installed on the device(s) that the Subscriber will connect to. Personal website certificates are typically installed in the certificate store of a web browser.

3) E-mail certificates

An e-mail certificate is installed to your e-mail application to enable secure e-mail communication. These certificates are known by many names such as e-mail security certificates, e-mail encryption certificates, S/MIME certificates, etc.

3.1 Mailbox Validated (MV) certificates

These are the standard certificates, where the customer only needs to demonstrate control of the e-mail address.

3.2 Organization Validated (OV) certificates

Includes only Organizational (Legal Entity) attributes in the Subject.

3.3 Sponsor Validated (SV) certificates

Combines Individual (Natural Person) attributes in conjunction with an Organizational (an associated Legal Entity) attribute.

3.4 Individual Validated (IV) certificates

Includes only Individual (Natural Person) attributes in the Subject.

4) Document Signing certificates

Document Signing certificates allow individuals, teams, and organizations to add an electronic, digital signature to a document in a variety of file formats to prove ownership.

5) Code Signing certificates

Code Signing certificates are used by software developers to digitally sign applications, drivers, executables, and software programs for end-users to verify that the code they receive has not been altered or compromised by a third party. Code Signing certificates are also used in secure software development to enforce continued integrity of the code base.

5.1 Organization Validated (OV) certificates

These are the standard certificates, requiring validation of the legitimacy of the organization.

5.2 Extended Validated (EV) certificates

EV certificates can be issued only by a subset of Certification Authorities (CAs) and require verification of the requesting entity's legal identity before certificate issuance.

Other types exist as well, such as business-to-business or machine-to-machine certificates.

In some documentation, certificate types are divided into classes, but this concept is vendor defined and not part of the PKI standard. For example (originally from VeriSign):

Class	Use	Level
1	Domain Validated (DV)	Low
2	Organization Validated (OV)	Substantial
3	Extended Validation (EV)	High
4	For online business transactions between companies	
5	For private organizations or governmental security	

Table 2: Certificate classes

In eIDAS terminology, the various types of Subscriber certificates are called Trust Services.

2.1.1 Single-purpose CAs

It is undesirable to issue multiple certificate types from the same Root, due to conflicting Certificate Policy or Trust Store requirements.

If several certificate types will be issued, determine first what this means for your hierarchy, and whether several Roots may be required to accommodate this. Extended Validation (EV) certificates (of any type) require their own Root.

Most Trust Stores strongly recommend or even require single-purpose CAs, so that each type of certificate is issued from its own Intermediate CA, as also shown in chapter 1.3.

2.2 Certificate Policy and Certification Practice Statement

There are two documents that must be created. Both should follow the template of RFC 7382[3]:

1) Certificate Policy (CP)

This document describes the policy for the entire hierarchy. There can be only 1 CP for the entire hierarchy, although Intermediate CAs may run stricter policies if they prefer.

[3] RFC 7382: https://www.rfc-editor.org/rfc/rfc7382

2) Certification Practice Statement (CPS)

This document describes the implementation of the above policy. Each CA (Root CA and any Intermediate CA) needs to create their own CPS. These practice statements do not necessarily have to be separate documents, but all services must be individually covered.

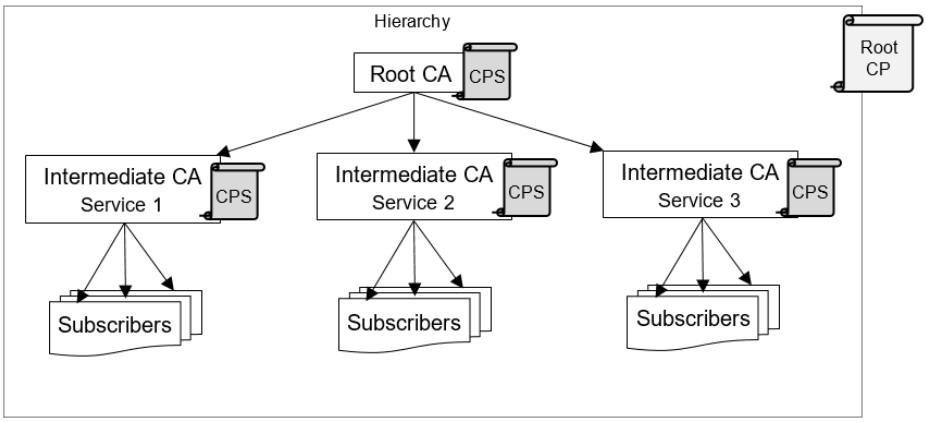

Figure 10: Relationship between CP and CPS

These together detail how the CA implemented the requirements and Best Practices to operate in a secure and trustworthy manner.

While the CP describes the options that *may be used*, and their minimum requirements, or stipulates that the CPS shall cover the details of items, it is important to realize that the CPS must describe exactly which of the options *are actually used and how*.

Each CPS describes the policy implementation for their own lower level certificates. So the Root CPS details how Intermediate certificates are handled, while the Intermediate CAs will talk about Subscriber certificates.

Examples of CP and CPS documents are easy to find, as Public Root CAs are required to publish theirs online on their websites[4].

If RFC 7382 is followed, the CP/CPS covers the following areas:

1) Introduction

Describes the CA itself, Overview, Document Name and Identification, PKI Participants, Certificate Usage, Policy Administration, and Definitions and Acronyms.

2) Publication and Repository Responsibilities

Describes Repositories, Publication of Certification Information, Time or Frequency of Publication, and Access Controls on Repositories.

[4] Thailand NRCA's CP and CPS: https://www.nrca.go.th/publish.html

3) Identification and Authentication

   Describes Naming, Initial Identity Validation, Identification and Authentication for Re-key Requests, and Identification and Authentication for Revocation Request.

4) Certificate Life Cycle Operational Requirements

   Describes Certificate Application, Certificate Application Processing, Certificate Issuance, Certificate Acceptance, Key Pair and Certificate Usage, Certificate Renewal, Certificate Re-key, Certificate Modification, Certificate Revocation and Suspension, and Certificate Status Services.

5) Facility, Management, and Operational Controls

   Describes Physical Controls, Procedural Controls, Personnel Controls, Audit Logging Procedures, Records Archival, Key Changeover, Compromise and Disaster Recovery, and CA or RA Termination.

6) Technical Security Controls

   Describes Key Pair Generation and Installation, Private Key Protection and Cryptographic Module Engineering Controls, Other Aspects of Key Pair Management, Activation Data, Computer Security Controls, Life Cycle Technical Controls, Network Security Controls, and Time-Stamping.

7) Certificate and CRL Profiles

   Describes the details of certificates and profiles, such as algorithms used, key sizes and other settings.

8) Compliance Audit and Other Assessments

   Describes how and how often audits are performed.

9) Other Business and Legal Matters

   Describes Fees, Financial Responsibility, Confidentiality of Business Information, Privacy of Personal Information, Intellectual Property Rights (if Applicable), Representations and Warranties, Disclaimers of Warranties, Limitations of Liability, Indemnities, Term and Termination, Individual Notices and Communications with Participants, Amendments, Dispute Resolution Provisions, Governing Law, Compliance with Applicable Law, and Miscellaneous Provisions.

Note that Best Practices are improved upon regularly, so the CP and/or CPS should be reviewed annually.

2.3 Infrastructure and security considerations

It has been a deliberate choice to order the activities such that the CP/CPS are created first, even if a formal, published, policy is not required if you are creating a Private (Internal) CA. Especially chapters 5 (Facility,

Management, and Operational Controls) and 6 (Technical Security Controls) will help you to make informed decisions and preparations for your environment.

2.3.1 RFC 3647

Following RFC 7382, chapters 5 and 6 refer to RFC 6484[5] chapters 5 and 6, which in turn refer to RFC 3647[6] chapters 4.5 and 4.6 for an overview of what kind of items must be addressed in the CP/CPS. In the following chapters, we will look at some of those, which require specific architectural planning (and may determine the cost of the operation).

2.3.1.1 (4.5.1) Physical Security Controls

In this subcomponent, the physical controls on the facility housing the entity systems are described. Topics addressed may include:

1) Site location and construction, such as the construction requirements for high-security zones and the use of locked rooms, cages, safes, and cabinets;
2) Physical access, i.e., mechanisms to control access from one area of the facility to another or access into high-security zones, such as locating CA operations in a secure computer room monitored by guards or security alarms and requiring movement from zone to zone to be accomplished using a token, biometric readers, and/or access control lists;
3) Power and air conditioning;
4) Water exposures;
5) Fire prevention and protection;
6) Media storage, for example, requiring the storage of backup media in a separate location that is physically secure and protected from fire and water damage;
7) Waste disposal; and
8) Off-site backup.

2.3.1.2 (4.5.2) Procedural Controls

In this subcomponent, requirements for recognizing trusted roles are described, together with the responsibilities for each role. Examples of trusted roles include system administrators, security officers, and system auditors.

For each task identified, the number of individuals required to perform the task (n out m rule) should be stated for each role. Identification and authentication requirements for each role may also be defined.

This component also includes the separation of duties in terms of the roles that cannot be performed by the same individuals.

[5] RFC 6484: https://www.rfc-editor.org/rfc/rfc6484

[6] RFC 3647: https://www.rfc-editor.org/rfc/rfc3647

2.3.1.3 (4.5.3) Personnel Security Controls

This subcomponent addresses the following:

1) Qualifications, experience, and clearances that personnel must have as a condition of filling trusted roles or other important roles. Examples include credentials, job experiences, and official government clearances that candidates for these positions must have before being hired;
2) Background checks and clearance procedures that are required in connection with the hiring of personnel filling trusted roles or perhaps other important roles; such roles may require a check of their criminal records, references, and additional clearances that a participant undertakes after a decision has been made to hire a particular person;
3) Training requirements and training procedures for each role following the hiring of personnel;
4) Any retraining period and retraining procedures for each role after completion of initial training;
5) Frequency and sequence for job rotation among various roles;
6) Sanctions against personnel for unauthorized actions, unauthorized use of authority, and unauthorized use of entity systems for the purpose of imposing accountability on a participant's personnel;
7) Controls on personnel that are independent contractors rather than employees of the entity; examples include:
   7.1 Bonding requirements on contract personnel;
   7.2 Contractual requirements including indemnification for damages due to the actions of the contractor personnel;
   7.3 Auditing and monitoring of contractor personnel; and
   7.4 Other controls on contracting personnel.
8) Documentation to be supplied to personnel during initial training, retraining, or otherwise.

2.3.1.4 (4.5.4) Audit Logging Procedures

This subcomponent is used to describe event logging and audit systems, implemented for the purpose of maintaining a secure environment. Elements include the following:

1) Types of events recorded, such as certificate lifecycle operations, attempts to access the system, and requests made to the system;
2) Frequency with which audit logs are processed or archived, for example, weekly, following an alarm or anomalous event, or whenever the audit log is n% full;
3) Period for which audit logs are kept;
4) Protection of audit logs:
   4.1 Who can view audit logs, for example only the audit administrator;
   4.2 Protection against modification of audit logs, for instance a requirement that no one may modify or delete the audit records or that only an audit administrator may delete an audit file as part of rotating the audit file; and
   4.3 Protection against deletion of audit logs.
5) Audit log back up procedures;

6) Whether the audit log accumulation system is internal or external to the entity;
7) Whether the subject who caused an audit event to occur is notified of the audit action; and
8) Vulnerability assessments, for example, where audit data is run through a tool that identifies potential attempts to breach the security of the system.

2.3.1.5 (4.6.2) Private Key Protection and Cryptographic Module Engineering Controls

Requirements for private key protection and cryptographic modules need to be considered for the issuing CA, repositories, subject CAs, RAs, and subscribers. For each of these types of entities, the following questions potentially need to be answered:

1) What standards, if any, are required for the cryptographic module used to generate the keys? A cryptographic module can be composed of hardware, software, firmware, or any combination of them. For example, are the keys certified by the infrastructure required to be generated using modules compliant with the US FIPS 140-1? If so, what is the required FIPS 140-1 level of the module? Are there any other engineering or other controls relating to a cryptographic module, such as the identification of the cryptographic module boundary, input/output, roles and services, finite state machine, physical security, software security, operating system security, algorithm compliance, electromagnetic compatibility, and self-tests.
2) Is the private key under n out of m multi-person control?(7) If yes, provide n and m (two person control is a special case of n out of m, where n = m = 2)?
3) Is the private key escrowed?(8) If so, who is the escrow agent, what form is the key escrowed in (examples include plaintext, encrypted, split key), and what are the security controls on the escrow system?
4) Is the private key backed up? If so, who is the backup agent, what form is the key backed up in (examples include plaintext, encrypted, split key), and what are the security controls on the backup system?
5) Is the private key archived? If so, who is the archival agent, what form is the key archived in (examples include plaintext, encrypted, split key), and what are the security controls on the archival system?
6) Under what circumstances, if any, can a private key be transferred into or from a cryptographic module? Who is permitted to perform such a transfer operation? In what form is the private key during the transfer (i.e., plaintext, encrypted, or split key)?
7) How is the private key stored in the module (i.e., plaintext, encrypted, or split key)?
8) Who can activate (use) the private key? What actions must be performed to activate the private key (e.g., login, power on, supply PIN, insert token/key, automatic, etc.)? Once the key is activated, is the key active for an indefinite period, active for one time, or active for a defined time period?
9) Who can deactivate the private key and how? Examples of methods of deactivating private keys include logging out, turning the power off, removing the token/key, automatic deactivation, and time expiration.
10) Who can destroy the private key and how? Examples of methods of destroying private keys include token surrender, token destruction, and overwriting the key.

11) Provide the capabilities of the cryptographic module in the following areas: identification of the cryptographic module boundary, input/output, roles and services, finite state machine, physical security, software security, operating system security, algorithm compliance, electromagnetic compatibility, and self-tests. Capability may be expressed through reference to compliance with a standard such as U.S. FIPS 140-1, associated level, and rating.

2.3.1.6 (4.6.5) Computer Security Controls

This subcomponent is used to describe computer security controls such as: use of the trusted computing base concept, discretionary access control, labels, mandatory access controls, object re-use, audit, identification and authentication, trusted path, security testing, and penetration testing. Product assurance may also be addressed.

A computer security rating for computer systems may be required. The rating could be based, for example, on the Trusted System Evaluation Criteria (TCSEC), Canadian Trusted Products Evaluation Criteria, European Information Technology Security Evaluation Criteria (ITSEC), or the Common Criteria for Information Technology Security Evaluation, ISO/IEC 15408:1999. This subcomponent can also address requirements for product evaluation analysis, testing, profiling, product certification, and/or product accreditation related activity undertaken.

2.3.1.7 (4.6.7) Network Security Controls

This subcomponent addresses network security related controls, including firewalls.

2.3.2 Other considerations

2.3.2.1 Enforce your hierarchical structure

Intermediate CA certificates should use the Path Length (`pathLenConstraint`) field in the Basic Constraints section. The Path Length field describes the maximum depth of a valid certification path. They are important as they describe the number of intermediate CA's in a hierarchy that need to be checked when checking the certificate path.

Where it appears, the number must be greater than or equal to zero. Where it does not appear, no limit is imposed. If it is 0, then no further Intermediate CAs are allowed and this Intermediate CA can only sign Subscriber certificates.

If an intermediate CA requires to sign another CA, the Path Length constraint should be set to its position in the CA hierarchy. For example, in a 3 tier hierarchy the Root CA path length will be unspecified, the next intermediate CA will have a length of one as it is required to sign another CA, and the last CA will have a value of zero.

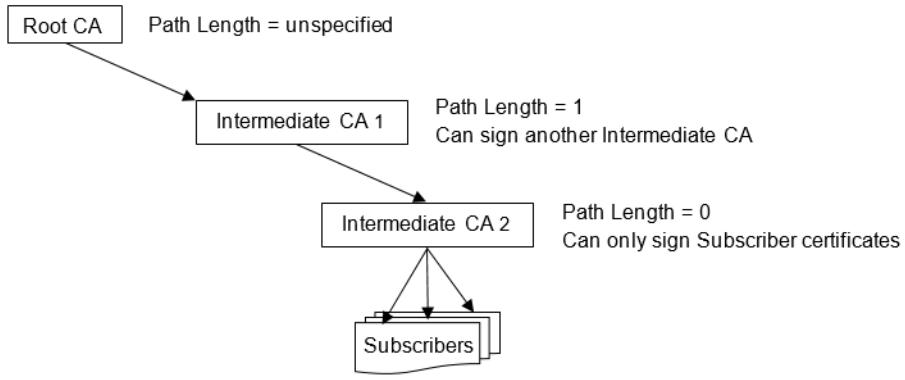

Figure 11: Using the Path Length Basic constraint

2.3.2.2 Protect your private keys

When a public/private key pair is generated, the key must be generated from a reliable source of randomness

A key pair should normally be generated by the Subscriber that will use it. If a private key has to be generated away from the Subscriber, then it must be encrypted in transit and at rest. Access to the key must be monitored, authenticated, and authorized. Note that the practice of generating keys for others is generally only allowed for Private (Internal) CAs, not for Public Root CAs.

Once a private key has been generated, it must be protected so that it can only be used by the identity it represents.

The public portion can be distributed to other users in the system. If a private key is compromised, an attacker could use it to impersonate the Subscriber and gain access to a system.

A CA's private key should be stored in hardware-based protection, such as a Hardware Security Module (HSM). This provides tamper-resistant secure storage. There are various vendors and models for these hardware solutions, internal and external, offering different security levels, and operating in a different way, so make sure that the Certificate Management System you have in mind supports the one you intend to use.

A private key for a Subscriber could be stored in a Trusted Platform Module (TPM) chip or a USB tamper-resistant security token.

2.3.2.3 Keep certificate lifetimes as short as practical

Having short certificate lifetimes for a Subscriber will reduce the impact of compromise, but increase the amount of administration overhead.

Root certificates should have longer lifetimes as they are the trust anchor for the entire PKI and renewing them requires all certificates in the chain to be reissued.

Intermediate certificates will have shorter lifetimes than the root certificate. Their exact duration should depend on security threats, administration overhead, system availability and the costs of issuing new intermediate CA certificates.

2.3.2.4 Keep the Root CA platform turned off

The Root CA platform is only needed if Intermediate certificates need to be created, updated, or revoked. To improve security, it is advised to keep the Root CA platform always turned off, until an action is needed involving an intermediate certificate.

Do not forget to make a backup of the certificate changes before powering down again.

2.4 Choosing a Certificate Management System

The choice of Certificate Management System (CMS) greatly determines which options and features are available to you. For example, the OpenSSL CA option shown in chapter 4 works well, but does not offer any process for things like an RA function or certificate distribution.

Each available CMS works differently (e.g., controlled on the command line, with a web browser, or server dialog boxes), there is no common advise for how relations and processes as described in chapter 1.2 could be set up.

An issued certificate can technically be valid longer than the Intermediate CA's certificate, but the verification will obviously fail. Depending on the CMS used, the creation or signing can fail, or it shortens the time the certificate is valid accordingly. This can be confusing and CA admins need to be aware of this and plan ahead, before any certificate is issued.

Note that certificates are not allowed to be deleted once published, for legal and trust reasons. They either expire or get revoked.

To recap from chapter 1.2, depending on your needs, the CMS may support the following roles:

1) The Registration Authority (RA) is the contact point for the Subscribers who want their certificate signed, renewed, suspended, or revoked. The RA can also play a role in the deployment of certificates.
2) The Certification Authority (CA) signs and revokes certificates, at the request of the Registration Authority.
3) The Validation Authority (VA) role is responsible for the availability of the CRL and optionally OCSP. OCSP is a specialized function and not always supported by a CMS. In that case, a third-party OCSP Responder may be needed.
4) The optional Auto Enrollment Gateway (AEG) is used for the automated enrollment, deployment and renewing of certificates. This may also be accomplished with a third-party solution.

Appendix A contains an overview of some of the available CMS platforms.

2.5 Certificate Transparency

Certificate Transparency[7] is an ecosystem that makes the issuance of SSL/TLS website certificates transparent and verifiable. To achieve this goal, all CAs are requested to log all newly created certificates into a set of publicly available, append-only, ledger sites. Because those are distributed and independent, anyone can query them to see which certificates have been included and when.

When someone submits a valid certificate to a log, the log responds with a Signed Certificate Timestamp (SCT), which is simply a promise to add the certificate to the log within some time period. The time period is known as the Maximum Merge Delay (MMD).

Although RFC 6962[8] says it is an experimental protocol, it is in fact mandatory for all Public Root CAs and is being enforced by an increasing number of web browsers these days, so make sure that your Certificate Management System supports it if you run a Public Root CA.

The issuance of certificates that are not CT compliant is not considered mis-issuance or a violation of protocols (it is not a requirement mandated by the CA/Browser Forum Baseline Requirements or the Trust Store policies as discussed in chapter 3.5, but mandated by web browser vendors); such certificates will simply fail to validate in CT-enforcing versions of modern browsers, as the web browsers cannot find proof in the ledgers that the certificates have actually been issued.

The policies of the relevant web browser vendors are linked in Appendix B.

2.5.1 Turning off CT for private domains

Since CT is typically not used within a Private CA, as that would expose details about your infrastructure, you will run into the problem that the SSL/TLS website certificates created within your Private CA hierarchy are not accepted by the latest versions of several of the major browsers, resulting in "CERTIFICATE_TRANSPARENCY_REQUIRED" error messages.

You will have to turn CT off for those internal domains in your devices.

Some guidance references are provided in Appendix B as well.

2.6 Distributing the root certificate

Any organization can technically become a CA or even a Root CA. Install the software, create the root certificate and you are good to go. Getting a CA up and running is only the first step though. You can now create SSL/TLS certificates, certificates for document signing, etc., but the world at large does not trust your

[7] Certificate Transparency: https://certificate-transparency.org

[8] RFC 6962: https://www.rfc-editor.org/rfc/rfc6962

certificates yet. Your website visitor's web browser will generate a warning that the presented certificate is not trustworthy.

For Private CAs, only used internally in an organization, that is not a problem. The created root certificate can be imported in the web browsers and Operating Systems throughout the organization (of course this can be automated).

2.7 Stay up-to-date with the industry

PKI will always be an evolving ecosystem, and new technologies, Best Practices, and both Trust Store and web browser policy updates appear regularly, all of which may have an immediate effect on the operation of your CA.

To stay up-to-date, it is advisable for all CA operators (including Private CAs) to regularly check, or even become a member of the CA/Browser Forum[9], which also hosts several working groups for different topics.

Public Root CAs may also be interested in joining the public discussion group of the Common CA Database (CCADB)[10].

2.8 When things go wrong

When a major event occurs, such as a critical vulnerability (2014, Heartbleed[11]), a bug in the Certificate Management System (2019, EJBCA[12]), or a breach of the CA, a CA may need to revoke a massive number of certificates, sometimes all of them.

Breaches are a well-known risk for all organizations, but the result of a breach of a CA can be devastating for both the organization and its Subscribers, given that the entire purpose of a CA is to offer trust.

There is a significant operational cost associated with mass revocations as well, in the form of a spike in Internet traffic of PKI-aware applications (such as web browsers) towards your published Certification Revocation List. Cloudflare described this cost in a blog[13], based on the Heartbleed vulnerability fallout.

CAs should prepare for any such disruption.

It is therefore that the CP, CPS, and the infrastructure plan should be drafted and implemented very carefully to both prevent incidents, and to limit the amount of damage an attacker can do.

---

[9] CA/Browser Forum: https://cabforum.org/

[10] CCADB Public: https://groups.google.com/a/ccadb.org/g/public

[11] Heartbleed: https://en.wikipedia.org/wiki/Heartbleed

[12] EJBCA bad serial numbers: https://www.thesslstore.com/blog/mass-revocation-millions-of-certificates-revoked-by-apple-google-godaddy/

[13] Heartbleed fallout: https://blog.cloudflare.com/the-hard-costs-of-heartbleed/

Specific care must also be taken at personnel awareness. Many of the breaches occur at the Registration Authority, where attackers gain access through phishing, allowing them to issue fraudulent certificates.

Setting up a Security Operations Center (SOC) to closely monitor your network, and a Computer Security Incident Response Team (CSIRT) to handle incidents, will help reduce incident detection and the amount of time to resolve them.

Appendix C contains an overview of several noteworthy failures.

2.8.1 Termination plan

There may come a time when it is no longer possible to continue operation of the CA. This could be for a variety of reasons, including an incident that is too impactful to recover from, or a bankruptcy.

Such unscheduled termination should be prepared for in a plan (eIDAS regulation requires this for Qualified Trust Service Providers, see chapter 5.2.1) and should at least cover which options your existing Subscriber base has for the continuity of their certificates. Can some of the CA services, such as the VA role, be taken over by another CA? ENISA published a set of guidelines[14] for this.

[14] Guidelines on Termination of Qualified Trust Services: https://www.enisa.europa.eu/publications/tsp-termination

3 Creating a Public Root CA

Creating a Public Root CA starts the same as any CA (so please start with chapter 2): establish the policy, build the infrastructure, and create the root certificate.

Please make sure to look at Certificate Transparency as well, as explained in chapter 2.5, which is a web browser requirement.

The creation of a Public Root certificate key pair is a very formal affair. After all, this is the trust anchor for the entire certificate chain. Creating trustworthiness for a Public Root CA requires several additional steps.

As mentioned in chapter 2.1, a Certificate Policy and Certification Practice Statement must be created. For a Public Root CA, those must also be published on the CAs website.

However, describing this in policy is not enough: it must be proven as well. Therefore, the creation of a root certificate key pair must be performed in an official ceremony, with independent witnesses and an audit to verify and document the success of the activity.

3.1 Resource protection with a CAA record in DNS

As per RFC 8659[15], "The Certification Authority Authorization (CAA) DNS Resource Record allows a DNS domain name holder to specify the Certification Authorities (CAs) authorized to issue certificates for that domain name. Publication of CAA Resource Records allows a public CA to implement additional controls to reduce the risk of unintended certificate mis-issue."

CAA records are one of the defense-in-depth layers. It leverages DNS as an information resource.

Checking CAA records is one of the steps in the domain validation process, performed by the Registration Authority, for all new Subscriber certificate requests.

The domain of a Public Root CA is used as a critical information publication resource, to publish documents such as the CP, CPS, and audit related information, as well as certificate status lookup services (CRL and/or OCSP), and some of those URIs also appear in issued certificates.

To protect the Public Root CA itself from certificates for its own domain being issued maliciously, it should publish a CAA record to state which CA (e.g., only itself or its Intermediate CAs) is allowed to issue such certificates.

[15] RFC 8659: https://www.rfc-editor.org/rfc/rfc8659

3.2 Quality assurance

Care must be taken to ensure that all certificates issued are valid and compliant with the standards and Best Practices.

This validation step is called linting. Depending on your Certificate Management System, this step can be performed either just before issuance of the certificate (on the Certificate Transparency pre-certificate if supported, see chapter 2.5), or after issuance.

Mozilla operates a helpful public tool[16] that can be used to perform post-issuance linting. It validates all certificates issued in a given date range against 3 popular open source liinters (x509_lint, CABLint and ZLint) and produces an overview of all problematic certificates that have thus been identified.

3.3 Additional requirements

There may also be additional requirements imposed depending on location (e.g., eIDAS regulation in the EU[17] - see chapter 5) and the selection of digital certificate types (Trust Services) offered.

Make sure to verify if the Certificate Management System you intend to use is compliant with all laws and regulations that are applicable to you.

3.4 Annual audit

Then an annual, formal audit is required by either WebTrust[18] or ETSI. Passing the audit can be viewed as the license to operate.

The requirements imposed on the Root CA are the same for all individual Intermediate CAs, or Subordinate CAs: they must also undergo annual audits.

3.5 Applying at Trust Stores

As mentioned in chapter 1.2.8.1, as a Public Root it is critical to get the root certificate added to the web browser and Operating System Trust Stores, as this is where Subscriber certificates are verified worldwide for validity and trustworthiness.

---

[16] CA Mis-issuance Checker: https://cachecker-dot-ccadb-231121.appspot.com/

[17] eIDAS Regulation: https://digital-strategy.ec.europa.eu/en/policies/eidas-regulation

[18] WebTrust: https://www.cpacanada.ca/en/business-and-accounting-resources/audit-and-assurance/overview-of-webtrust-services

All these parties are members of the CA/Browser Forum. Organized in 2005, [they] are a voluntary group of certification authorities (CAs), vendors of Internet browser software, and suppliers of other applications that use X.509 v.3 digital certificates for SSL/TLS, code signing, and S/MIME[19].

One of the services provided by the CA/Browser Forum is a Common CA Database (CCADB)[20]. This database is used to maintain the status of each Public Root CA and its Intermediate CAs, including the latest versions of their CP, CPS, and audit statements.

While technical requirements published in RFCs are inherently static in nature, the CA/Browser Forum closely follows and decides on innovation and technological progress, including the security and privacy domains. Two key Best Practices document they maintain are called CA/Browser Forum Baseline Requirements, one for SSL/TLS certificate issuers[21] and one for S/MIME certificate issuers[22], which set forth the minimum requirements to be addressed in the CP.

All web browser and Operating System vendors list this set of Baseline Requirements (BR) as a mandatory minimum addition to the basic requirements that are audited by WebTrust and ETSI, to get considered for inclusion to their Trust Store. Some Trust Stores impose additional requirements on top of this baseline. These policies can be found in Appendix B. There are other Trust Stores, such as Android and Ubuntu/Debian, but without stated policy website. Those seem to automatically follow Mozilla's list of approved Roots.

Another requirement is providing broad benefit to the Internet community; CAs with a limited user base do not provide enough incentive for inclusion in the Trust Stores.

Each application has several phases. The first phase is for the Trust Store to verify if all technical requirements have been met. This is followed by a complete inspection of the published CP, CPS, and audit documents. Once all requirements have been found in order, a public discussion can be held with the CA/Browser Forum community to decide whether the applying Root CA is trustworthy and will indeed bring additional value.

[19] CA/Browser Forum: https://cabforum.org/

[20] CCADB: https://www.ccadb.org/

[21] CA/Browser Forum Baseline Requirements (SSL/TLS Server Certificates) : https://cabforum.org/baseline-requirements-documents/

[22] CA/Browser Forum Baseline Requirements (S/MIME Certificates) : https://cabforum.org/smime-br/

4 Example: creating an Internal CA with OpenSSL

The example in this chapter creates a Private (Internal) CA based on OpenSSL[23], which is one of the default packages available in most Linux distributions. One can be set up fairly quickly and it well explains the relationship between the various PKI infrastructure components. Once ready, it can be used to create server and user SSL/TLS certificates. We will build the Chain of Trust from the ground up, as shown in chapter 1.4.

4.1 Preparation

4.1.1 Install required packages

First we need to make sure we have the required packages installed.

Check if we have OpenSSL installed:

```
$ openssl version -a
```

If not, install it with the following command

```
$ sudo apt install openssl
```

Install the Network Time Protocol (NTP) package:

```
$ sudo apt install ntp
```

You can see the list of NTP servers your machine synchronizes with. Run this command to see the list.

```
$ ntpq -p
```

4.2 Create the Root CA

4.2.1 Create the directories

It is best for maintenance and security to organize our new CA with dedicated folders, so we create some directories and files required to set up the CA.

Let us first switch to superuser mode:

```
$ sudo su
```

Create the root directory for the CA (under the home directory of the superuser, 'root').

```
# mkdir /root/ca
```

[23] OpenSSL: https://www.openssl.org/

Create four more directories underneath the 'ca' directory:

1) A '`certs`' directory to store issued certificates.
2) A '`crl`' directory to store the Certificate Revocation List.
3) A directory '`newcerts`' to store new certificates.
4) A directory '`private`' to store private keys.

```
# cd /root/ca
# mkdir certs crl newcerts private
# chmod 700 private
```

Create an empty file '`index.txt`' which will serve as a flat-file database for issued certificates.

```
# touch index.txt
```

Create a file named '`serial`' which stores the serial number of the next certificate.

```
# echo 1000 > serial
```

Note that the value 1000 is a hexadecimal number, which is 4096 in decimal format.

4.2.2 Create the configuration file

We need a configuration file for OpenSSL to use. Copy the configuration file from the default installation:

```
# cp /usr/lib/ssl/openssl.cnf .
```

Open the configuration file using any text editor application, for example with nano:

```
# nano openssl.cnf
```

The `[ ca ]` section is mandatory. Here we tell OpenSSL to use the options from the `[ CA_default ]` section.

```
[ ca ]
# `man ca`
default_ca = CA_default
```

Scroll down the file to the `[ CA_default ]` section. Update the CA path.

```
[ CA_default ]
# Directory and file locations.
dir               = /root/ca
certs             = $dir/certs
crl_dir           = $dir/crl
new_certs_dir     = $dir/newcerts
database          = $dir/index.txt
serial            = $dir/serial
RANDFILE          = $dir/private/.rand

# The root key and root certificate.
private_key       = $dir/private/ca.key.pem
certificate       = $dir/certs/ca.cert.pem

# For certificate revocation lists.
```

```
crlnumber          = $dir/crlnumber
crl                = $dir/crl/ca.crl.pem
crl_extensions     = crl_ext
default_crl_days   = 30

# SHA-1 is deprecated, so use SHA-2 instead.
default_md         = sha256

name_opt           = ca_default
cert_opt           = ca_default
default_days       = 375
preserve           = no
policy             = policy_strict
```

We apply `policy_strict` for all Root CA signatures, as the Root CA is only being used to create Intermediate CAs.

```
[ policy_strict ]
# The root CA should only sign intermediate certificates that match.
# See the POLICY FORMAT section of `man ca`.
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional
```

We apply `policy_loose` for all Intermediate CA signatures, as the Intermediate CA will be used for signing server and subscriber certificates that may come from a variety of parties.

```
[ policy_loose ]
# Allow the intermediate CA to sign a more diverse range of certificates.
# See the POLICY FORMAT section of the `ca` man page.
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional
```

Options from the `[ req ]` section are applied when creating certificates or Certificate Signing Requests.

```
[ req ]
# Options for the `req` tool (`man req`).
default_bits        = 2048
distinguished_name  = req_distinguished_name
string_mask         = utf8only

# SHA-1 is deprecated, so use SHA-2 instead.
default_md          = sha256

# Extension to add when the -x509 option is used.
x509_extensions     = v3_ca
```

The `[ req_distinguished_name ]` section declares the information normally required in a Certificate Signing Request. You can optionally specify some defaults.

```
[ req_distinguished_name ]
# See <https://en.wikipedia.org/wiki/Certificate_signing_request>.
countryName                     = Country Name (2 letter code)
stateOrProvinceName             = State or Province Name
localityName                    = Locality Name
0.organizationName              = Organization Name
organizationalUnitName          = Organizational Unit Name
commonName                      = Common Name
emailAddress                    = Email Address

# Optionally, specify some defaults.
countryName_default             = TH
stateOrProvinceName_default     = Bangkok
localityName_default            =
0.organizationName_default      = ETDA
#organizationalUnitName_default =
#emailAddress_default           =
```

The next few sections are extensions that can be applied when signing certificates. For example, passing the `-extensions v3_ca` command-line argument will apply the options set in `[ v3_ca ]`.

We will apply the `v3_ca` extension when we create the root certificate.

```
[ v3_ca ]
# Extensions for a typical CA (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true
keyUsage = critical, digitalSignature, cRLSign, keyCertSign
```

We will apply the `v3_ca_intermediate` extension when we create the Intermediate Certificate. `pathlen:0` ensures that there can be no further Certification Authorities below the Intermediate CA.

```
[ v3_intermediate_ca ]
# Extensions for a typical intermediate CA (`man x509v3 config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true, pathlen:0
keyUsage = critical, digitalSignature, cRLSign, keyCertSign
```

We will apply the `usr_cert` extension when signing client certificates, such as those used for remote user authentication.

```
[ usr_cert ]
# Extensions for client certificates (`man x509v3_config`).
basicConstraints = CA:FALSE
nsCertType = client, email
nsComment = "OpenSSL Generated Client Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
keyUsage = critical, nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = clientAuth, emailProtection
```

We will apply the `server_cert` extension when signing server certificates, such as those used for web servers.

```
[ server_cert ]
# Extensions for server certificates (`man x509v3_config`).
basicConstraints = CA:FALSE
nsCertType = server
nsComment = "OpenSSL Generated Server Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer:always
keyUsage = critical, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
```

The `crl_ext` extension is automatically applied when creating Certificate Revocation Lists.

```
[ crl_ext ]
# Extension for CRLs (`man x509v3 config`).
authorityKeyIdentifier=keyid:always
```

We will apply the `ocsp` extension when signing the Online Certificate Status Protocol (OCSP) certificate.

```
[ ocsp ]
# Extension for OCSP signing certificates (`man ocsp`).
basicConstraints = CA:FALSE
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
keyUsage = critical, digitalSignature
extendedKeyUsage = critical, OCSPSigning
```

Save the file with Ctrl+o. and exit with Ctrl+x.

4.2.3 Create the Root Certificate

First we need to create the private key for our root certificate. Run the following OpenSSL command to create an RSA private key `ca.key.pem` with a length of 4096 bits:

```
# openssl genrsa -aes256 -out private/ca.key.pem 4096
```

It will ask to set a passphrase. Choose a strong passphrase and make sure to record it in a safe place.

Keep the file secure

```
# chmod 400 private/ca.key.pem
```

Now we can create the root certificate using the private key from the previous step. Run the following command to create a certificate that expires in 3650 days (about 10 years):

```
# openssl req -config openssl.cnf -new -x509 -key private/ca.key.pem -
days 3650 -sha256 -extensions v3_ca -out certs/ca.cert.pem
```

It asks for several details, including the passphrase of the private key. Enter all the details to generate the root certificate. Keep a record of the details, because the same information is required to create Certificate Signing Requests (CSRs) for all other certificates.

```
Country Name (2 letter code) [XX]:TH
State or Province Name (full name) []:Bangkok
Locality Name (eg, city) []:Bangkok
Organization Name (eg, company) []:ETDA
Organizational Unit Name (eg, section) []:IT-Security
Common Name (e.g. server FQDN or YOUR name) []:ETDA CA
Email Address []: info@etda.or.th
```

Make the file read-only.

```
# chmod 444 private/ca.cert.pem
```

All done, let us see what the root certificate looks like.

```
# openssl x509 -noout -text -in certs/ca.cert.pem
```

The Issuer and Subject are identical as the certificate is self-signed. Note that all root certificates are self-signed.

```
Signature Algorithm: sha256WithRSAEncryption
    Issuer: C=TH, ST=Bangkok,
            O=ETDA, OU=IT-Security,
            CN=ETDA CA
    Validity
        Not Before: Jan 22 12:22:58 2023 GMT
        Not After : Jan 10 12:22:58 2033 GMT
    Subject: C=TH, ST=Bangkok,
             O=ETDA, OU=IT-Security,
             CN=ETDA CA
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            Public-Key: (4096 bit)
```

The output also shows the X509v3 extensions. We applied the `v3_ca` extension, so the options configured in `[ v3_ca ]` should be reflected in the output.

```
X509v3 extensions:
    X509v3 Subject Key Identifier:
        3C:5A:29:2F:6B:57:79:4F:39:FD:32:35:63:74:92:60:6E:E8:2A:47
    X509v3 Authority Key Identifier:
        keyid:3C:5A:29:2F:6B:57:79:4F:39:FD:32:35:63:74:92:60:6E:E8:2A:47

    X509v3 Basic Constraints: critical
        CA:TRUE
    X509v3 Key Usage: critical
        Digital Signature, Certificate Sign, CRL Sign
```

The Root Certificate can now be deployed to all devices (e.g., web browsers) in your internal network, as explained in chapter 2.6.

4.3 Create the Intermediate CA

It is technically possible to use the Root CA directly to create Subscriber certificates, without an Intermediate CA, or Subordinate CA, in between. This is a dangerous option, as explained earlier, so we will now create an Intermediate certificate.

You can setup several Intermediate CAs if you want to separate the services offered (e.g., issuing server and client certificates is performed by different departments) in a 2-tier installation, or if you want to set up a multi-tier hierarchy, as shown in chapter 1.3.

4.3.1 Create the directories

The Intermediate CA is ideally located on a different server, so that the Root CA can be kept turned off once we are finished creating the Intermediate CA, as explained in chapter 2.3.2.4.

For this example, we will create our Intermediate CA in a subdirectory of the Root CA.

```
# mkdir /root/ca/intermediate
```

The same files and directories as for the Root CA are needed, with the addition of a '`csr`' subdirectory where we will store Certificate Signing Requests.

```
# cd /root/ca/intermediate
# mkdir certs crl csr newcerts private
# chmod 700 private
# touch index.txt
# echo 1000 > serial
```

Add a `crlnumber` file to the intermediate CA directory tree, to keep track of Certificate Revocation Lists.

```
# echo 1000 > /root/ca/intermediate/crlnumber
```

4.3.2 Create the configuration file

The configuration is almost the same as for the Root CA, so make a copy of that first

```
# cp /root/ca/openssl.cnf .
# nano openssl.cnf
```

And make the following 5 changes

```
[ CA_default ]
dir             = /root/ca/intermediate
private_key     = $dir/private/intermediate.key.pem
certificate     = $dir/certs/intermediate.cert.pem
crl             = $dir/crl/intermediate.crl.pem
policy          = policy_loose
```

If you are installing a multi-tier hierarchy, you will also need to update the `pathlen` variable in the `v3_ca_intermediate` configuration item of each Intermediate CA tier, as described in chapter 2.3.2.1.

4.3.3 Create the Intermediate Certificate

The same operation as for the Root Certificate. First create the private key:

```
# openssl genrsa -aes256 -out private/intermediate.key.pem 4096
# chmod 400 private/intermediate.key.pem
```

It will ask to set a passphrase. Choose a strong passphrase and make sure to record it in a safe place.

Then we create the Intermediate certificate itself. This is done with a Certificate Signing Request to the Root CA. The details should generally match the Root CA, but the Common Name must be different.

```
# cd /root/ca/intermediate
# openssl req -config openssl.cnf -new -sha256 -key
private/intermediate.key.pem -out csr/intermediate.csr.pem
```

To create an Intermediate certificate, use the Root CA with the `v3_intermediate_ca` extension to sign the Intermediate CSR. The Intermediate certificate should be valid for a shorter period than the Root certificate, for example half of it (so about 5 years).

```
# cd /root/ca
# openssl ca -config openssl.cnf -extensions v3_intermediate_ca -days
1825 -notext -md sha256 -in intermediate/csr/intermediate.csr.pem -out
intermediate/certs/intermediate.cert.pem
```

(If the Root CA is located on a different server, this step involves copying the Intermediate CSR to the Root CA, generating the certificate with the shown command, and then copying the Intermediate certificate back to the Intermediate CA).

As we did for the root certificate, check that the details of the intermediate certificate are correct.

```
# openssl x509 -noout -text -in intermediate/certs/intermediate.cert.pem
```

Verify the Intermediate certificate against the Root certificate. An OK indicates that the Chain of Trust is intact.

```
# openssl verify -CAfile certs/ca.cert.pem
intermediate/certs/intermediate.cert.pem

intermediate.cert.pem: OK
```

4.3.4 Create the certificate chain file

When an application (e.g., a web browser) tries to verify a certificate signed by the Intermediate CA, it must also verify the Intermediate certificate against the Root certificate. To complete the Chain of Trust, create a CA certificate chain to present to the application.

To create the CA certificate chain, concatenate the Intermediate and Root certificates together (the order does not matter). We will use this file later to verify certificates signed by the Intermediate CA.

```
# cat intermediate/certs/intermediate.cert.pem certs/ca.cert.pem >
intermediate/certs/ca-chain.cert.pem
```

```
# chmod 444 intermediate/certs/ca-chain.cert.pem
```

If you deployed the Root Certificate to all devices in your internal network, then the chain file only needs to contain the Intermediate certificate.

Our CA is now ready to start issuing Subscriber Certificates.

4.4 Sign a server or client certificate

The first step depends on the Subscriber who needs a signed certificate. They can ask you to either create the certificate for them (including the private key!), or they create one themselves and send a Certificate Signing Request (CSR) file to you.

The verification of the CSR is performed by the Registration Authority, while the actual signing of the certificate is performed by the Certification Authority.

In this example we want a certificate for www.example.com

4.4.1 Create the certificate

First create the private key. Our Root and Intermediate pairs are 4096 bits. Server and client certificates normally expire after one year, so we can safely use 2048 bits instead, as is common on the Internet.

```
# cd /root/ca/intermediate
# openssl genrsa -aes256 -out private/www.example.com.key.pem 2048
# chmod 400 private/www.example.com.key.pem
```

Then create the Certificate Signing Request with

```
# openssl req -config openssl.cnf -key private/www.example.com.key.pem -
new -sha256 -out csr/www.example.com.csr.pem
```

The CSR details do not need to match the Intermediate CA.

- For server certificates, the Common Name must be a fully qualified domain name (e.g., www.example.com).
- For client certificates it can be any unique identifier (e.g., an e-mail address).

4.4.2 Sign the certificate

To create a certificate, we use the Intermediate CA to sign the CSR.

- If the certificate is going to be used on a server, use the `server_cert` extension.
- If the certificate is going to be used for user authentication, use the `usr_cert` extension.

Certificates are usually given a validity of one year, though a CA will typically give a few days extra for convenience.

```
# cd /root/ca/intermediate
# openssl ca -config openssl.cnf -extensions server_cert -days 375 -
notext -md sha256 -in csr/www.example.com.csr.pem -out
certs/www.example.com.cert.pem
# chmod 444 certs/www.example.com.cert.pem
```

Verify the certificate

```
# openssl x509 -noout -text -in certs/www.example.com.cert.pem
```

The Issuer is the Intermediate CA. The Subject refers to the certificate itself.

The output will also show the X509v3 extensions. When creating the certificate, you used either the `server_cert` or `usr_cert` extension. The options from the corresponding configuration section will be reflected in the output.

Using the CA certificate chain file we created earlier (`ca-chain.cert.pem`), we can verify that the new certificate has a valid Chain of Trust.

```
# openssl verify -CAfile certs/ca-chain.cert.pem
certs/www.example.com.cert.pem

www.example.com.cert.pem: OK
```

4.4.3 Deploy the certificate

You can now either deploy your new certificate to a server, or distribute the certificate to a Subscriber. When deploying to a server application (e.g., Apache), you need to make the following files available:

- `ca-chain.cert.pem`
- `www.example.com.key.pem`
- `www.example.com.cert.pem`

Note that, as an Internal CA, we do not use Certificate Transparency, so you will need to also update your web browsers to exclude this domain from CT checking, as explained in chapter 2.5.1.

If you are only signing a CSR from the Subscriber (i.e., did not create the certificate), you do not have access to their private key so you only need to give them back the chain file (`ca-chain.cert.pem`) and the certificate (`www.example.com.cert.pem`).

4.5 Revoke a certificate

If an issued certificate needs to be revoked, this is done with the following command.

```
# cd /root/ca/intermediate
# openssl ca -config openssl.cnf -revoke certs/www.example.com.cert.pem
```

4.6 Certificate Revocation Lists (CRLs)

A certificate revocation list (CRL) provides a list of revoked certificates. A client application, such as a web browser, can use a CRL to check a server's authenticity. A server application, such as Apache or OpenVPN, can use a CRL to deny access to Subscribers that are no longer trusted.

Publish the CRL at a publicly accessible location (e.g., http://example.com/intermediate.crl). Third parties can fetch the CRL from this location to check whether any certificates they rely on have been revoked.

CRL files are usually published with filename extension `.crl` (rather than `.crl.pem`) and should be served on a plain HTTP (no HTTPS) URI with Content-type "`application/pkix-crl`".

Note that each level in the CA hierarchy has its own CRL. The Root CA only signs Intermediate certificates, but those can be revoked as well. For internal use, it may not be necessary to create a formal CRL for the Root CA. Otherwise, repeat the process from this chapter for the Root CA.

4.6.1 Create the configuration file

When a Certification Authority signs a certificate, it will normally encode the CRL location into the certificate. Add `crlDistributionPoints` to the appropriate sections. In our case, add it to the previously created `[ server_cert ]` section.

```
[ server_cert ]
crlDistributionPoints = URI:http://example.com/intermediate.crl
```

4.6.2 Create the CRL

```
# cd /root/ca/intermediate
# openssl ca -config openssl.cnf -gencrl -out crl/intermediate.crl
```

You can check the contents of the CRL with the `crl` option

```
# openssl crl -in crl/intermediate.crl -noout -text

No Revoked Certificates
```

You should recreate the CRL regularly. By default, the CRL expires after 30 days. This is controlled by the `default_crl_days` option in the `[ CA_default ]` section.

If a certificate was signed with an extension that includes `crlDistributionPoints`, a client-side application can read this information and fetch the CRL from the specified location.

The CRL distribution points are visible in the certificate X509v3 details.

```
# openssl x509 -in myblog.example.com.cert.pem -noout -text

    X509v3 CRL Distribution Points:

        Full Name:
          URI:http://example.com/intermediate.crl
```

4.7 Online Certificate Status Protocol (OCSP)

The Online Certificate Status Protocol (OCSP) was created as an alternative to Certificate Revocation Lists (CRLs) as described in the previous chapter. Like CRLs, OCSP enables a requesting party (e.g., a web browser) to determine the revocation state of a certificate.

When a CA signs a certificate, they will typically include an OCSP server address (e.g., http://ocsp.example.com) in the certificate. This is similar in function to `crlDistributionPoints` used for CRLs.

When a web browser is presented with a server certificate, it will send a query to the OCSP server address specified in the certificate. At this address, an OCSP responder listens to queries and responds with the revocation status of the certificate.

4.7.1 Create the configuration file

To use OCSP, the CA must encode the OCSP server location into the certificates that it signs. Use the `authorityInfoAccess` option in the appropriate sections, which in our case means the previously created `[ server_cert ]` section.

```
[ server_cert ]
authorityInfoAccess = OCSP;URI:http://ocsp.example.com
```

4.7.2 Create the OCSP Certificate

Create a private key and encrypt it with AES-256 encryption.

```
# cd /root/ca/intermediate
# openssl genrsa -aes256 -out private/ocsp.example.com.key.pem 4096
```

Create a certificate signing request (CSR). The details should generally match those of the signing CA, but the Common Name must be a fully qualified domain name (e.g., ocsp.example.com).

```
# openssl req -config openssl.cnf -new -sha256 -key
private/ocsp.example.com.key.pem -out csr/ocsp.example.com.csr.pem
```

Sign the CSR with the Intermediate CA.

```
# openssl ca -config openssl.cnf -extensions ocsp -days 375 -notext -md
sha256 -in csr/ocsp.example.com.csr.pem -out
certs/ocsp.example.com.cert.pem
```

Verify that the certificate has the correct X509v3 extensions.

```
# openssl x509 -noout -text -in certs/ocsp.example.com.cert.pem

    X509v3 Key Usage: critical
        Digital Signature
    X509v3 Extended Key Usage: critical
        OCSP Signing
```

4.7.3 Running an OCSP Responder

Important note: While the OpenSSL package includes a tool `ocsp` to run an OCSP Responder, it is only meant for testing purposes and an OCSP Responder from another vendor should be used for production.

Run the OCSP responder on localhost. Rather than using the revocation status from a separate CRL file, the (test!) OCSP responder reads the index.txt file directly. The response is signed with the OCSP cryptographic pair (using the `-rkey` and `-rsigner` options).

```
# cd /root/ca/intermediate
# openssl ocsp -port 127.0.0.1:2560 -text -sha256 -index index.txt -CA
certs/ca-chain.cert.pem -rkey private/ocsp.example.com.key.pem -rsigner
certs/ocsp.example.com.cert.pem -nrequest 1
```

5 eIDAS and (Qualified) Trust Service Providers

eIDAS (**e**lectronic IDentification, **A**uthentication, and trust **S**ervices)[24] is an EU regulation on electronic identification and trust services for electronic transactions in the European Single Market. It was established in EU Regulation 910/2014 of 23 July 2014 on electronic identification and repeals 1999/93/EC from 13 December 1999.

The eIDAS Regulation:

- ensures that people and businesses can use their own national electronic identification schemes (eIDs) to access public services available online in other EU countries;
- creates a European internal market for trust services by ensuring that they will work across borders and have the same legal status as their traditional paper-based equivalents.

Only by providing certainty on the legal validity of these services will businesses and citizens use digital interactions naturally.

The EU terminology for a general Public CA is Trust Service Provider, a Public Root CA is called a Qualified Trust Service Provider.

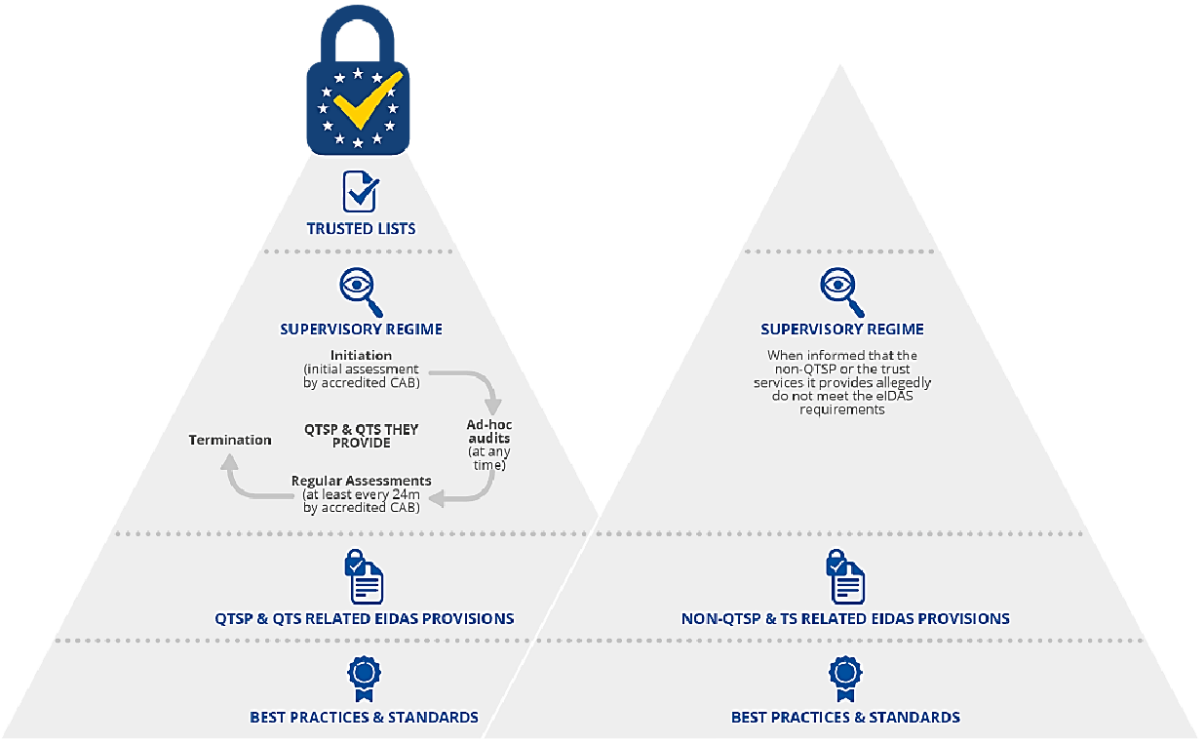

Figure 12: Trust Service Providers and Qualified Trust Service Providers

[24] eIDAS regulation: https://digital-strategy.ec.europa.eu/en/policies/eidas-regulation

5.1 Trust Service Providers

Under the EU eIDAS regulation the definition of a trust service is:

An electronic service normally provided for remuneration which consists of:

1) the creation, verification, and validation of electronic signatures, electronic seals or electronic timestamps, electronic registered delivery services and certificates related to those services, or
2) the creation, verification, and validation of certificates for website authentication; or
3) the preservation of electronic signatures, seals or certificates related to those services.

TSPs (Trust Service Providers) provide a combination of the above services – some only issue digital certificates, others provide electronic signature services as well. What is important is that all TSPs adhere to the strict requirements that ensure the validity and security of the certificates, keys, and signatures they provide.

TSPs must operate to a set of standards to ensure the security and validity of the certificates and authentication services they offer. The EU's eIDAS regulation has helped standardize the requirements for TSPs and provide organizations with a list of European Commission approved companies that they can trust.

5.1.1 Security Framework for TSPs

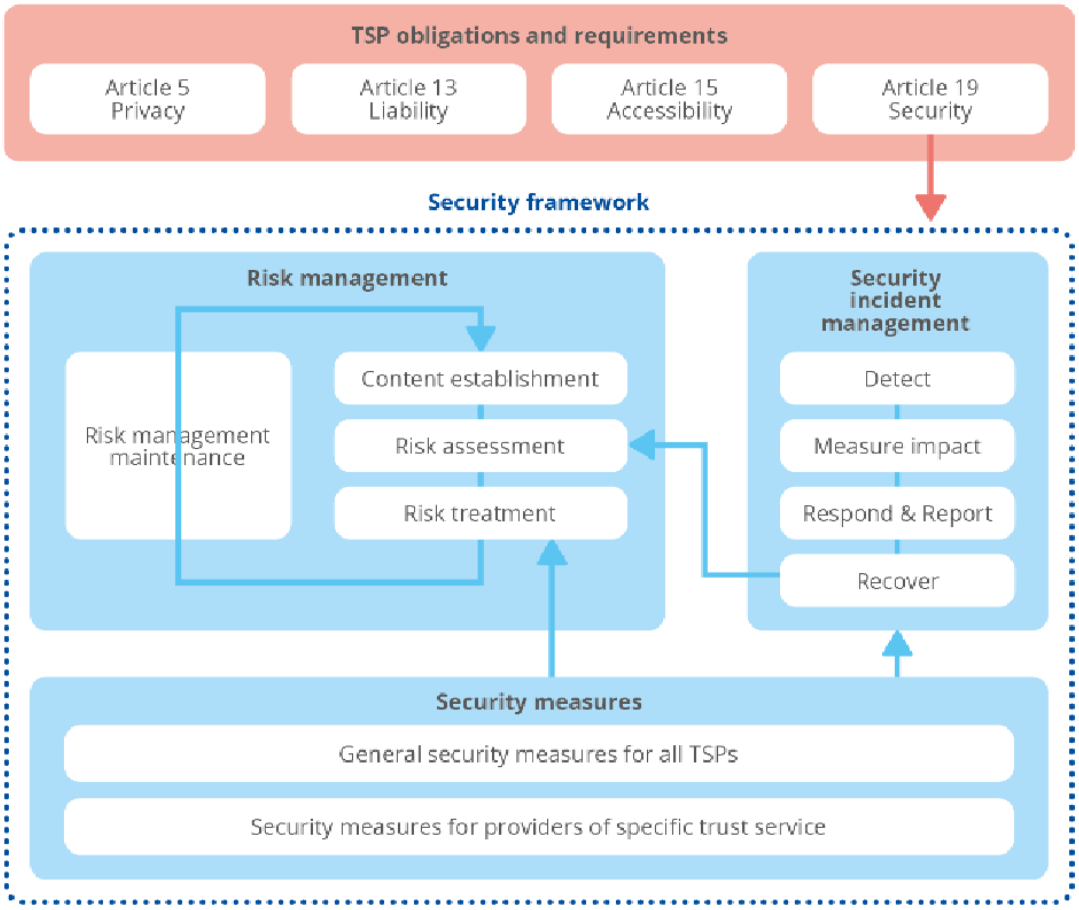

Figure 13: Security Framework for Trust Service Providers

ENISA provides a document to describe the security framework requirements[25]:

1) Risk management related to the security of the eIDAS trust services and based on ISO/IEC 27005 general approach;
2) Security incident management by using the appropriate measures to efficiently detect, measure the impact, respond, report, and recover from security incidents as part of the eIDAS Regulation;
3) Security measures recommended to TSPs from "technical" standards and best practices to treat the risks and contribute to the security incident management. The level of security of these measures is to be selected by the TSP to be commensurate to the degree of risk bound to the context of the TSP (determined during the "context establishment").

[25] Security Framework for Trust Service Providers : https://www.enisa.europa.eu/publications/security-framework-for-trust-providers

5.2 Qualified Trust Service Providers

A Qualified Trust Service Provider (QTSP) must comply with additional measures under the eIDAS regulation to provide qualified certificates, qualified electronic signatures, qualified electronic seals, or qualified electronic signature creation devices.

To be listed and recognized as a Qualified Trust Service Provider, organizations must undergo an independent assessment and regular audits to ensure that they continue to adhere to the QTSP requirements set out by eIDAS. The additional requirements are to ensure that the integrity of the data held by QTSPs for the creation of digital certificates and signing keys is secure and protected to ensure validity.

Once an organization has submitted the required information and QTSP status is awarded, the organization can be listed as a QTSP on the European Council's list of trusted providers.

5.2.1 Security Framework for QTSPs

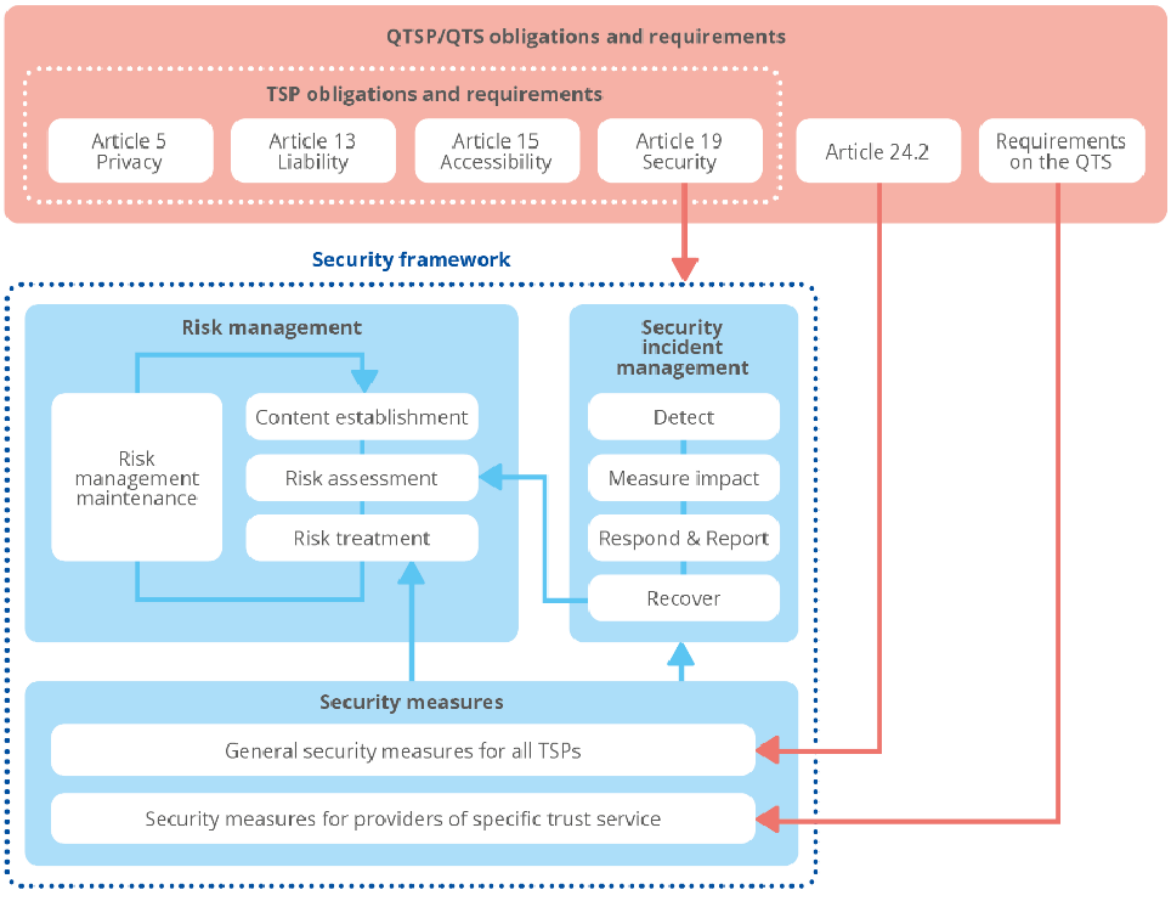

Figure 14: Security Framework for Qualified Trust Service Providers

ENISA provides a document to describe the security framework requirements[26]:

The eIDAS Regulation introduced the notions of qualified trust service (QTS) and qualified trust service provider (QTSP) with a view to indicating requirements and obligations that ensure high-level security and a higher presumption of their legal effect. Because of this higher presumption of legal effects (e.g. Qualified Electronic Signature, QES, has the equivalent legal effect to handwritten signature), the consequences of a security incident can have a higher impact on a QTSP than those of a non-QTSP.

Focus areas in addition to the requirements for all TSPs are:

Risk Management:

1) Level of security required by the qualified status
   1.1 As the Impact increases, the degree of risk also increases and so the level of security required for QTSP may be higher than one for non-QTSP.
2) Presumption of negligence
   2.1 'Guilty until proven innocent', requires strict documentation and procedures.
3) Conformity assessments
   3.1 Regular ISO and WebTrust audits.
4) Measures against forgery and theft of data

Security Incident Management:

1) Termination plan
   1.1 This document shall be verified because of its particular importance regarding the sustainability and durability of QTSs and to boost users' confidence in the continuity of qualified trust services, such as in exceptional/unfortunate cases of QTSP unscheduled termination (e.g., bankruptcy).

Security measures:

1) Depending on the services offered, higher levels of security are required for the creation and maintenance of digital certificates and signatures.

[26] Security Framework for Qualified Trust Service Providers:
https://www.enisa.europa.eu/publications/security-framework-for-qualified-trust-providers

6 eIDAS 2.0: European digital identity (eID)

In October 2020, the European Council called for the development of an EU-wide framework for secure public electronic identification (eID), including interoperable digital signatures, to provide people with control over their online identity and data as well as to enable access to public, private and cross-border digital services.

The Commission presented a proposal for a regulation on the European eID in June 2021. The proposal includes the creation of a European digital wallet app that allows for safe storage, management and sharing of documents.

The Council adopted its position for negotiations on eID on 6 December 2022[27]. The Council wants to make sure the European digital wallet is highly secure, respectful of data privacy and free for everyone.

Operated via digital wallets available on mobile phone apps and other devices to:

1) identify online and offline
2) store and exchange information provided by governments e.g. name, surname, date of birth, nationality
3) store and exchange the information provided by trusted private sources
4) use the information as confirmation of the right to reside, to work, or to study in a certain Member State

6.1 Practical use

The European Digital Identity can be used for any number of cases, for example:

1) public services such as requesting birth certificates, medical certificates, reporting a change of address
2) opening a bank account
3) filing tax returns
4) applying for a university, at home or in another Member State
5) storing a medical prescription that can be used anywhere in Europe
6) proving your age
7) renting a car using a digital driving license
8) checking in to a hotel

[27] A digital future for Europe: https://www.consilium.europa.eu/en/policies/a-digital-future-for-europe/

6.2 Certification

The implementation path has been described as well[28].

The regulation should leverage, rely on, and mandate the use of relevant and existing Cybersecurity Act certification schemes, or parts thereof, to certify the compliance of wallets, or parts thereof, with the applicable cybersecurity requirements.

Consequently, the Cybersecurity Act framework applies fully, including the peer review mechanism between national cybersecurity certification authorities provided within the Cybersecurity Act.

To align the eID regulation and the Cybersecurity Act to the extent possible, member states will designate public and private bodies accredited to certify the wallet as provided in the Cybersecurity Act.

[28] Implementation: https://www.consilium.europa.eu/en/press/press-releases/2022/12/06/european-digital-identity-eid-council-adopts-its-position-on-a-new-regulation-for-a-digital-wallet-at-eu-level/

Appendix A: Certificate Management Systems

This appendix lists some of the available Certificate Management Platforms. Please refer to chapter 2.4 for a description of the roles in the checkmark columns.

Name	Manufacturer	CA	VA	RA	AEG	Primary Control	Restrictions
Open Source							
Dogtag Certificate System	Endi S. Dewata	✓	✓	✓		Web browser	
EJBCA Community Edition	Keyfactor	✓	✓	✓		Web browser	
gnoMint	Davefx	✓	✓	✓		X-Windows	SSL/TLS
OpenSSL	The OpenSSL Project	✓	✓			Command line	SSL/TLS
XCA	Christian Hohnstaedt	✓	✓			X-Windows	
XiPKI	XiPKI	✓	✓	✓		Command line	SSL/TLS
Commercial							
Atlas	GlobalSign	✓	✓	✓	✓	Web browser	
DigiCert PKI Platform	DigiCert	✓	✓	✓	✓	Web browser	
EJBCA Enterprise Edition	Keyfatcor	✓	✓	✓	✓	Web browser	
Entrust Certificate Authority	Entrust	✓	✓	✓	✓	Windows or X-Windows	
Microsoft Certification Authority	Microsoft	✓	✓	✓	✓	Windows Server	

Table 3: Some of the available Certificate Management Systems

Appendix B: References

Standards and Best Practices

Publisher	Document	
ETSI	Overview of all related ETSI standards	Link
ETSI	EN 319 401 V2.3.1 (2021-05): Electronic Signatures and Infrastructures (ESI); General Policy Requirements for Trust Service Providers	Link
ETSI	EN 319 411-1 V1.3.1 (2021-05): Electronic Signatures and Infrastructures (ESI); Policy and security requirements for Trust Service Providers issuing certificates; Part 1: General requirements	Link
ETSI	EN 319 411-2 V2.4.1 (2021-11): Electronic Signatures and Infrastructures (ESI); Policy and security requirements for Trust Service Providers issuing certificates; Part 2: Requirements for trust service providers issuing EU qualified certificates	Link
ETSI	TR 119 411-4 V1.1.1 (2018-05): Electronic Signatures and Infrastructures (ESI); Policy and security requirements for Trust Service Providers issuing certificates; Part 4: Checklist supporting audit of TSP against ETSI EN 319 411-1 or ETSI EN 319 411-2	Link
ENISA	Security Framework for Trust Service Providers	Link
ENISA	Security framework for Trust Service Providers - Technical Guidelines on Trust Services	Link
ENISA	Security Framework for Qualified Trust Service Providers	Link
ENISA	Recommendations for QTSPs based on Standards - Technical guidelines on trust services	Link
ENISA	Trust Service Provider Technical Best Practices - Considering the EU eIDAS Regulation (910/2014)	Link
ENISA	Guidelines on Termination of Qualified Trust Services	Link
RFC	RFC 3647: Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework	Link
RFC	RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile	Link
CAB	CA/Browser Forum Baseline Requirements Certificate Policy for the Issuance and Management of Publicly-Trusted Certificates (*SSL*/TLS Server Certificates)	Link
CAB	CA/Browser Forum Baseline Requirements for the Issuance and Management of Publicly-Trusted S/MIME Certificates	Link
CAB	CA/Browser Forum Network and Certificate System Security Requirements	Link

Regulation (Europe)

Publisher	Document	
European Commission	eIDAS Regulation	Link
European Commission	Learn about eIDAS	Link
European Commission	eIDAS 2.0	Link

Templates

Publisher	Document	
RFC	RFC 7382: Template for a Certification Practice Statement (CPS) for the Resource PKI (RPKI)	Link
GlobalSign	Template Certification Practice Statement	Link

Audit policies

Publisher	Document	
WebTrust	WebTrust for Certification Authorities Principles and Criteria	Link
CCADB	Common CCADB Policy	Link

Trust Store policies

Publisher	Document	
Microsoft	Program Requirements - Microsoft Trusted Root Program	Link
Mozilla	Mozilla's CA Certificate Program	Link
Apple	Apple Root Certificate Program	Link
Google	Chrome Root Program	Link
Adobe	Adobe Approved Trust List	Link

Certificate Transparency browser policies

Publisher	Document	
Apple	Apple's Certificate Transparency Policy	Link
	More information (including how to turn in off for selected internal domains)	Link
Google	Chrome Certificate Transparency Policy	Link
	More information (including how to turn in off for selected internal domains)	Link

Appendix C: Noteworthy Certification Authority failures

Comodo (2011)

An attacker by compromised several Comodo resellers and obtained rogue certificates for www.google.com, mail.google.com, addons.mozilla.org, login.live.com, login.yahoo.com, and login.skype.com. Comodo was trusting resellers to perform domain control validation, which is a critical certificate authority function, instead of doing it themselves.
https://blog.mozilla.org/security/2011/03/25/comodo-certificate-issue-follow-up/

DigiNotar (2011)

An unknown attacker compromised DigiNotar and after obtaining full administrative access to all critical CA systems, issued rogue certificates for numerous domains. A rogue wildcard certificate for google.com was used for mass interception of traffic from Iranian citizens. The Fox-IT investigators dubbed the incident "Operation Black Tulip", and their report identified 300,000 Iranian Gmail accounts as the main victims of the hack. After more than 500 fake DigiNotar certificates were found, major web browser makers reacted by blacklisting all DigiNotar certificates, resulting in severe economic damage to the country. The scale of the incident was used by some organizations like ENISA and AccessNow.org to call for a deeper reform of HTTPS in order to remove the weakest link possibility that a single compromised CA can affect that many users.
https://www.enisa.europa.eu/media/news-items/operation-black-tulip/

CNNIC (2015)

CNNIC violated their own Certification Practice Statement, and issued an unconstrained Intermediate CA certificate to an external organization with no Certification Practice Statement or technical infrastructure whatsoever to operate a certificate authority. That organization subsequently used the Intermediate CA to forge certificates for Google and likely other domains.
https://security.googleblog.com/2015/03/maintaining-digital-certificate-security.html

Let's Encrypt (2015)

ACME, the automated issuance protocol used by Let's Encrypt, suffers from a cryptographic flaw that would allow attackers to fraudulently obtain certificates for domains they do not control. The flaw had gone undetected during a formal security audit. Fortunately, the flaw was discovered and fixed before Let's Encrypt went live.
https://www.agwa.name/blog/post/duplicate_signature_key_selection_attack_in_lets_encrypt

Symantec (2015)

Over a period of several years, Symantec issued over 100 test certificates for 76 different domains without the authorization of the domain owners. This was discovered when Google's Certificate Transparency log monitor detected an unauthorized certificate for google.com in Certificate Transparency logs.
https://security.googleblog.com/2015/10/sustaining-digital-certificate-security.html

Symantec (2017)

In March 2017, Google and Mozilla engineers found that Symantec mis-issued 127 SSL certificates, but as the investigation progressed this initial estimation grew to a whopping figure of over 30,000 certs.
https://www.bleepingcomputer.com/news/security/google-outlines-ssl-apocalypse-for-symantec-certificates/

MonPass (2021)

Hackers have breached a server belonging to MonPass, one of Mongolia’s largest certificate authorities (CA), and have backdoored the company’s official client with a Cobalt Strike-based backdoor.
https://therecord.media/mongolian-certificate-authority-hacked-eight-times-compromised-with-malware/

Appendix D: Other real world uses of PKI

The Internet itself is also vulnerable to attacks. To increase the level of security, several protocol additions have been developed that also use PKI. We will quickly introduce some of them.

IPsec

IPsec as a standard for virtual private networking (VPN), which depends upon PKI via the Internet Key Exchange (IKE) extension[29].

RPKI for BGP

The Border Gateway Protocol (BGP) is the protocol almost exclusively used in the Internet to exchange routing information between network domains.

Resource Public Key Infrastructure (RPKI)[30] has been added to BGP to prevent from route hijacking.

DNSSEC

DNS Security Extensions (DNSSEC)[31] is described in a set of RFCs[32] and is designed to authenticate DNS response data. It verifies responses to ensure a DNS server's response is what the zone administrator intended. It does not address all threats (nothing does), but it provides a building block for providing additional data security, and not just within the DNS but also within the applications and services that are built on it.

Note, that DNSSEC is not only for the Web, but also can be used by any other Internet service or protocol. We are already seeing interesting uses of DNSSEC with email (SMTP), instant messaging and voice-over-IP.

DANE

DNS-Based Authentication of Named Entities (DANE)[33] allows you to securely specify exactly which TLS/SSL certificate an application or service should use to connect to your site. If, for instance, a web browser supporting DANE detects that it is not using the specified certificate, it can warn you that your connection is insecure… even though you see a "lock" icon. Beyond the web, DANE is being used with remarkable success in securing communication over email and over instant messaging protocols such as XMPP (Jabber). There is also interest in using DANE for SIP (voice-over-IP (VoIP)) and other communication methods[34].

---

[29] RFC 6071: https://www.rfc-editor.org/rfc/rfc6071

[30] RFC 7454: https://www.rfc-editor.org/rfc/rfc7454

[31] DNSSEC: https://www.internetsociety.org/deploy360/dnssec/

[32] DNSSEC RFCs: https://www.internetsociety.org/resources/deploy360/2011/dnssec-rfcs-3/

[33] RFC 6698: https://www.rfc-editor.org/rfc/rfc6698

[34] DANE: https://www.internetsociety.org/resources/deploy360/dane/