POLITECNICO DI TORINO

Master degree course in Computer Engineering (Cybersecurity)

Master Degree Thesis

Securing digital identities: from the deployment to the analysis of a PKI ecosystem with virtual HSMs leveraging open-source tools

Supervisor
prof. Antonio Lioy

Candidate
Alessandro Loconsolo

Academic year 2023-2024

† To my grandmother
Giuseppina

Summary

The objective of this work is to implement a PKI using exclusively open-source tools, with a particular focus on the integration of EJBCA CE with a virtual HSM. The primary goal is to establish trust between the Authorities and the End Entities within the ecosystem and then to identify the principal challenges that might be encountered during the deployment, from a security, performance and management point of view.

The system has been implemented within a containerised environment, with Docker Compose orchestrating the modules of the infrastructure. The designed architecture comprises two EJBCA CE instances, which have been configured as the CA and the VA, respectively. Each instance is equipped with its own database and virtual HSM, and its functionalities are subject to RBAC, thereby ensuring that the principle of least privilege is upheld for the entities within the organisation. SoftHSM2 has been selected as the HSM for this project, due to its strong compatibility with the EJBCA framework.

The analysis of the resulting PKI demonstrates that such open-source tools offer a viable and cost-effective option for the provision of strong authentication, particularly in terms of scalability and flexibility. From a performance perspective, the infrastructure has shown the capacity to issue a significant number of certificates with high throughput. Nevertheless, the system exhibits several security deficiencies, including the inability to segregate the CA from the RA, the incompatibility of EJBCA CE with more modern and secure enrolment protocols, such as ACME and EST, and the utilisation of a virtual HSM, which cannot be considered a substitute for a physical one in any case.

In conclusion, this solution is suitable for a variety of applications, for instance educational purposes or testing environments. In the context of production, this solution may be adopted by SMBs that do not have complex requirements and wish to enhance their security posture with a minimal investment. Conversely, large enterprises, that have more rigorous security demands, and must adhere to strict compliance policies and establish public trust, will find this solution unsuitable for their needs.

Acknowledgements

I would like to thank prof. Antonio Lioy, who supported me throughout my thesis project. His expert feedback and advice played a key role in elevating the quality of this work.

I also wish to express my gratitude to prof. Miguel Ángel Solinas, of Universidad Nacional de Córdoba, who has proposed me this project to me during my mobility year in Argentina. His guidance in the initial stages of my thesis allowed me to set a solid foundation for my research.

Furthermore, I would like to thank my colleagues at my new (and first) job as a Security Analyst at iCubed, especially the more experienced ones. Their teachings provided precious insights into cybersecurity and gave me a taste of how a real production environment is. Even though I joined the team during the final phase of my thesis, their contributions have greatly enriched my project and boosted my professional growth.

Then, I want to express my deepest gratitude to Argentina and Argentines, because I have discovered a wonderful country with a vibrant culture. Despite being thousands of kilometres away from my loved ones, those people always made me feel at home, sharing unforgettable and intense moments that I will always keep in my hearth.

In addition, I would like to thank Politecnico di Torino and the city of Torino. Reflecting on my university career evokes a mix of emotions; it was a tough journey, marked by challenges and setbacks, yet also rich with opportunities and moments of joy. Along the way, I met incredible people and had experiences that not only strengthened me but also played a crucial role in shaping the person I am today.

To all my friends, I am deeply grateful for your support and the joyful moments we've shared. You have all contributed to making me a better person.

I dedicate this achievement to my family. To my mother, who has always believed in me and encouraged me never to give up, even in the hardest moments. I will forever take your courage as an example throughout my life. To my father, I am eternally grateful for your sacrifices and the dedication you've shown in your work, which inspires me to strive for excellence in all I do.

To my grandmother Giuseppina, who recently passed away, thank you for raising me to value the simple yet profound virtues of life. With little, you gave me the world, teaching me to remain humble and to cherish what truly matters. Your wisdom will always guide me every day. To my other grandparents Alessandro, Giovanna and Saverio, I am sure you all are watching over me from above and you are celebrating with me this award.

Finally, to my other half, Chiara, whose affection and support have been my anchor. Your presence brightens my days and inspires me to chase my dreams. Thank you for being my constant companion and source of encouragement.

Contents

Chapter 1

Introduction

In an era where the digital realm is exponentially increasing [1], the average Internet user can be involved in a wide range of activities, each varying in levels of significance and security implications. On one end of the spectrum, there are basic tasks such as conducting a simple web search, akin to leafing through a magazine, or scrolling through social media feeds, comparable to the casualness of enjoying a beer with friends. While these activities contribute to an individual's digital footprint, they often carry minimal weight in terms of personal identification or security implications. Conversely, on the other end, there are digital actions that mirror the signing of a contract or the exchange of currency in the physical world. These include online banking, e-commerce transactions, and the exchange of legally binding documents. Here, the stakes are significantly higher as they involve financial commitments, contractual obligations, and legal consequences. Just as one's identity and intent are meticulously verified when signing a physical contract or making a bank transaction in person, similar verification is crucial in the digital space.

People's reliance on Information Systems grows, hence the necessity for robust security measures increase as well. Among these, for instance, there is Public Key Infrastructure (PKI) that secures digital communications and validates identities online, ensuring that the entities involved in transactions are authentic, and the information exchanged remains secure and unaltered. The importance of such defense strategies has been further emphasised in 2020 by the impact of the Covid-19 pandemic, which has dramatically transformed workplace dynamics. In this circumstance, PKI, along with VPN, played a key role in granting the shift towards remote work, providing a secure and scalable solution for the companies against cyber attacks [2].

The primary objective of this Master's Degree Thesis Project is to implement a X.509 PKI ecosystem using open-source frameworks, taking into account security and performance standards, and subsequently conduct a comprehensive study of its behaviour. The motivation behind this work is that this technology is so widespread and, although it is far from perfect, it is still the most effective approach to providing a secure authentication [3]. PKI is, in fact, a relatively old technology; the concept of certificate can be traced back to the late 1970s, but its true value has only been recognised in the last two decades. Indeed, at its very beginning, the framework displayed a number of complications that made it inadequate in addressing the real-world needs for digital identity verification and security, and there were several detractors that considered this technological ecosystem useless and obsolete [4]. However, PKI's core principles remained vital for ensuring the integrity and security of online transactions, so despite the initial challenges and after several adjustments, it is now widely recognised as one of the fundamental components for a solid security infrastructure. This is confirmed by the fact that the PKI market was valued at USD 3032.9 million in 2022 and is expected to increase at a compound annual growth rate (CAGR) of 20.6% in the time window 2023-2032 (Fig. 1.1) [5].

The reasons of PKI's success lays on its flexibility to seamlessly integrate emerging technologies, such as updating algorithms, into its structure; this adaptability makes it easily implemented in many fields like Cloud Infrastructures, Healthcare, IoT etc. On the other hand there are several drawbacks and weaknesses to monitor, i.e. the lack of trust models with worldwide acceptance

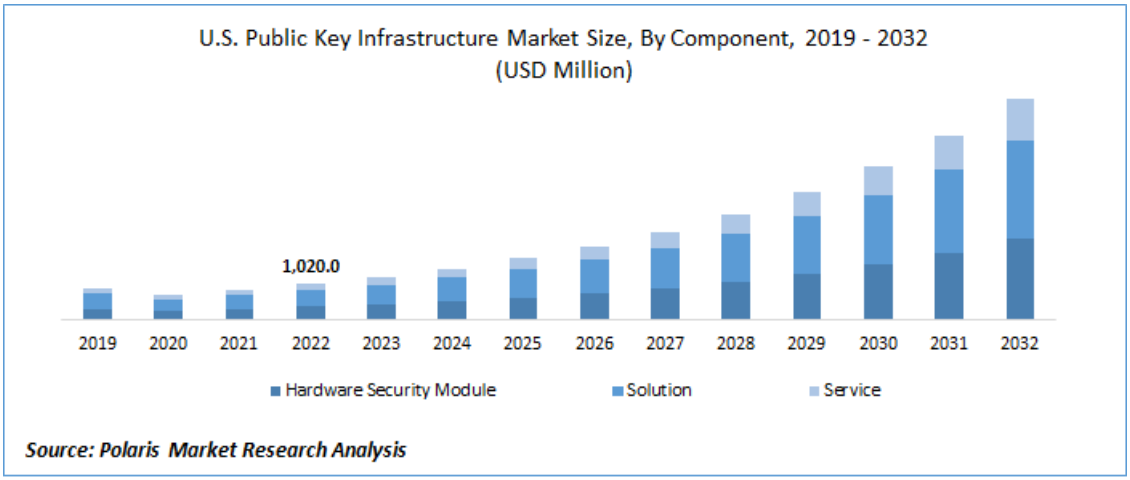

Figure 1.1: PKI Market Trend (source: Polaris).

or scalability challenges. Furthermore, there are a couple of threats not to underestimate: internal ones, where the lack of cooperation among stakeholders undermines trust and endangers the infrastructure, and external threats, such as attacks targeting the CA infrastructure (Table 1.1).

Strenghts	**Weaknesses**
Flexibility to accomodate and assimilate technologies	Absence of Globally anchored-trust models and cross-country certifications Usability and Scalability Issues
Opportunities	**Threats**
Infrastructure – Cloud New domains – IoT	Cohesiveness among Stakeholders Exploitation of CA Infrastructure

Table 1.1: SWOT Analysis of PKI ecosystem (source: [6]).

This work will utilise several open-source tools as instrumental case studies, such as Docker, EJBCA Community Edition and SoftHSM, and afterwards will evaluate the functionality, security and efficiency of the designed system. A key aspect of this analysis is the assessment of the security framework of the PKI, that involves examining the cryptographic processes, key management strategies, and the overall robustness of the security measures implemented. The use of Docker, that containerizes the system, enhances replicability and consistency of the whole environment.

EJBCA provides a practical perspective on Certificate Authority management and serves as a real-world example of how PKI components actually interact together. SoftHSM is the software implementation of a Hardware Security Module and allows to perform key management and cryptographic operations within a simulated, yet realistic, secure environment.

Finally, this project aims to identify the critical issues that might arise in the deployment and operation of PKI in a production environment, highlighting potential vulnerabilities, performance bottlenecks, and scalability challenges, questioning if implementations of this kind can actually be viable in a production context. The outcomes of this analysis will result in formulating recommendations and best practices for the deployment of PKI systems, particularly emphasizing the aspects that require careful consideration and revision before implementation in a real-world scenario.

This document is structured as follows, in Chapter 2 all the key concepts of PKI will be reviewed; then Chapter 3 will introduce the EJBCA Framework, explaining its functionalities and the design choices for this project; Chapter 4 will illustrate how the virtual HSM is integrated inside the ecosystem; in Chapter 5 the best practices to maintain a PKI environment will be shown, along with an analysis of the robustness, and in general of the performance, of the designed infrastructure; finally, the conclusions of this work will be presented in Chapter 6, which will determine the most appropriate uses for the implemented system.

Chapter 2

Background

2.1 Public Key Infrastructure

2.1.1 PKI defined

Nowadays, even the smallest company needs a security infrastructure, which brings a wide range of benefits such as an easier access to security for applications, a more enhanced login process, and a user friendly transparency. Furthermore, this approach ensures comprehensive protection throughout the system that is, cost reduction, easier management, effective safeguarding and lets the businesses choose providers with deep expertise in this filed.

According to Adams Carlisle and Lloyd Steve in "Understanding PKI: Concepts, Standards, and Deployment Considerations" [7, Chap. 3],

> A Public Key Infrastructure (PKI) is the basis of a pervasive security infrastructure whose services are implemented and delivered using public-key concepts and techniques.

In detail, the three indispensable services that a PKI should provide are:

- *Authentication*: provides the certainty to one party that another is truly who they claim to be. This is achieved by means of a digital signature.
- *Confidentiality*: guarantees that only the intended recipient(s) can access specific data, ensuring its privacy. To obtain this one, data encryption is needed. The most common example of PKI for confidentiality purposes is in the context of TLS.
- *Integrity*: ensures that data remains unchanged, whether in transit or over time, safeguarding it from both intentional and accidental alterations. Integrity is obtained, like authentication, with a digital signature, or through a Message Authentication Code (MAC).

There are many other components, functions and services a PKI can provide, as illustrated in Table 2.1. However, it is important to understand that an environment might not aim to incorporate all these elements, instead, it may select only the ones that are specifically tailored to its needs.

Several of these aspects will be analysed more closely in the subsequent sections.

2.1.2 Asymmetric cryptography

At the foundation of PKI lies *asymmetric cryptography*. In this ciphering model an entity has a key-pair consisting of a secret component (SK) which must be kept protected, and the public key (PK) which is accessible to other parties. Such couplet may fulfil two functions:

Certification Authority	Certificate Repository	Certificate Revocation
Key Backup	Key Recovery	Automatic Key Update
Key History Management	Cross-Certification	Client Software
Authentication	**Integrity**	**Confidentiality**
Secure Time Stamping	Notarization	Non-Repudiation Support
Secure Data Archive	Privilege/Policy Creation	Privilege/Policy Verification

Table 2.1: All the services a PKI can offer (in bold the core ones).

- *Encrypting/Decrypting data*: the sender uses the receiver's PK to encrypt some information that can be then decrypted only using the corresponding SK.
- *Digital Signature*: a person can sign a document using their SK, and then another entity can validate the authenticity of that document using the corresponding PK.

By the way, there are some aspects to consider, although asymmetric cryptography is more secure than symmetric ciphers, where only one key is generated and all the involved parties must share the same secret (with all the associated problems), it cannot always be employed due to its heavy computational requirements. Hence if a document is several megabytes large, the best practice is to encrypt it using a symmetric key, which is faster, then encrypt this one, that usually has a size of few kilobytes, with public key cryptography and finally send the whole data to the receiver.

Another factor to take into account is the security of the key-pair, as it is technically feasible to derive the secret from the public key. In fact, this kind of algorithms are extremely vulnerable if the key size is too small, hence the challenge is balancing robust protection with system performance. Larger keys, while offering stronger security, can lead to increased computational load and slower processing times. With current technology, an algorithm like RSA 2048 may offer a suitable compromise, however in the next decade, the advent of quantum computing will likely render such key sizes insufficient, and the adoption of more advanced cryptographic methods, namely post-quantum cryptography, will be necessary.

The key-pair generation requires complex algorithms and often a Random Number Generation. The most common algorithms are DSA, RSA, DH and the elliptic curve ciphers, like ECDSA or ECDH. Then, the secret key must be protected:

- during storage, because an unauthorized duplication of the private key could enable an entity to impersonate its rightful owner.
- during usage, when the private key is utilised for operations such as digital signing or decryption, it must be provided to a CPU. If the system is infected by some malware, the key will be compromised.

The easiest way to generate a key-pair is using a software application, but it could be dangerous since computers may be infected by malwares and there is an higher risk to create weak keys, if the algorithm of for the generation is not properly implemented. Another way is using some dedicated hardware, like a smart-card, but the drawback is that it is really difficult to develop a hardware update in case a vulnerability patch is needed.

A third solution for the key-pair generation could be generating the keys using a well built software and then inject the private key inside the hardware secure device.

2.1.3 Non-Repudiation

Non-repudiation is an important service in PKI. This ensures that a user cannot disclaim a specific action, since it is cryptographically bound to that individual, hence any potential denial of that operation constitutes an admission of malice or negligence. There exist several variants:

- *non-repudiation of origin*
- *non-repudiation of receipt*
- *non-repudiation of creation*
- *non-repudiation of delivery*
- *non-repudiation of approval*

A Public-Key Certificate that accomplishes this property can be actually be used as a proof into a court in case of a disguise or claim with the other party. Anyway, not all systems are required to enable it, for instance for casual e-mail and messages between two friends or Web browsing using a TLS server authentication, ensuring authentication, integrity and confidentiality may be sufficient, while in case of financial transactions, contracts and agreements or online voting, it is important to guarantee that the entities cannot deny their actions.

Non-repudiation is a characteristic unique to public-key cryptography and cannot be effectively implemented on a symmetric key infrastructure, because in case two entities share a symmetric key there is a potential for a dispute. For instance, if one party creates and encrypts a receipt, and then sends it to its counterpart, the recipient could falsely claim to have originated the document, as both have access to the shared key. This situation creates a loophole where either entity could deny their actions by attributing them to the other. Instead, in a PKI-enabled system, this issue does not exist anymore because the receipt would be signed using the private key, only accessible by the creator, unless it has been compromised by other means.

In a PKI, non-repudiation could be achieved only if other services are enabled. Such dependencies are:

- *Secure Time-Stamping*: consists in a secure association of time with data, performed by a trusted authority that guarantees its authenticity and integrity. Sometimes an explicit time representation is not necessary; a sequential number indicating the document's order of presentation to the authority may be sufficient. However it is crucial that the parties can confirm the authenticity and integrity of the time stamp linked to the document. This service is really important to enhance non-repudiation because it provides the evidence that a specific event occurred at a specific time.
- *Notarization*: in a PKI environment, this service, facilitated by an entity called notary, certifies the validity or correctness of a piece of data by means of mechanisms like signature verification and checking validity of the public key. It enforces the non-repudiation because provides a verifiable proof of the data's integrity and origin.
- *Secure Archival*: For potential dispute resolution, it's essential to securely archive evidence such as expired certificates, outdated Certification Revocation Lists (CRLs), time-stamp tokens, data certification structures, and similar data. The database must be cryptographically protected using a digital signature to ensure data integrity.
- *Human Factor*: Although the above services give an important contribute to obtain non-repudiation, an eventual dispute resolution will almost always need a human judgement.

Non-repudiation is the most difficult and complex service, a well designed system should provide all the means useful to make a proper decision in case of a claim. For this reason it is not correct that a PKI can provide non-repudiation, but rather it can be said that it supports non-repudiation, i.e. creates, maintains, and archives some of the evidence that will be needed when dispute resolution is performed.

2.2 X.509 certificate

2.2.1 Public-Key Certificate

A Public-Key Certificate (PKC) is a data structure to securely bind a public key to some attributes that identify the owner of the corresponding private key.

Typically, securely binding is obtained with the signature by a Trusted Third Party (TTP), usually a Certification Authority (CA), but other methods exist, such that blockchains, direct trust (e.g. TLS) and personal signature. There exist several types of certificates but the most common are the X.509 ones, particularly version 3, which will be the focus of examination in this project.

The attributes are related to the private key's owner, and since there are multiple ways to identify an entity, they may vary. For transaction protection, it is important to consider only the ones that are relevant to that specific operation. Often, choosing the meaningful attributes may be challenging due to their significant variability and because they are not known a priori.

When a PKC will be used for legal matters, such as buying or selling things or signing a contract, enabling non-repudiation is strictly required. Public-Key Certificate represents the public component of the corresponding personal private key.

2.2.2 The X.509 standard

X.509 is the most widely adopted standard for creating certificates, it was invented in 1988 by ITU (International Telecommunication Union), but the first version was not really successful. Version 3, released in 1996, is considered an efficient solution to the problem of identifying the owner of a cryptographic key, and its flexibility has lead to its broad acceptance as a standard format, facilitating interoperability among various tools and applications. The structural syntax of an X.509 certificate is based on ASN.1 (Abstract Syntax Notation One)

There are several sources that document the technical aspects of X.509 certificate, the most relevant are the RFC (Request for Comments), a series of notes published by Internet Engineering Task Force (IETF) and by Internet Society (ISOC), and the Recommendation X.509, by ITU.

As described in RFC-5280 [8, Sec. 4.1], an X.509 certificate has several fields which can be either mandatory or optional (figure 2.1):

- *Version* of the certificate. If v1 value is 0, if v2 value is 1, if v3 value is 2.
- *Serial Number*: unique identifier for this certificate with respect to the certificate issuer.
- *Signature*: algorithm identifier, i.e. Object Identifier (OID), which indicates with a code the cipher used to calculate the digital signature of the certificate.
- *Issuer*: contains the general information, called Distinguished Name (DN), which involve Common Name (CN) Organization (O) and Country (C) of the Certification Authority that has issued the certificate.
- *Validity*: represents the window of time that this certificate should be considered valid unless otherwise revoked.
- *Subject*: indicates the certificate owner's DN.
- *Subject Public Key Info*: is the public key and algorithm identifier associated to the certificate.
- Issuer Unique ID and Subject Unique ID are optional fields which are not recommended for use.

The *Extensions* field is optional and includes additional information useful to better identify the owner of the certificate. This is is a fundamental element that has significantly contributed to the success and widespread adoption of the X.509v3 model because it introduced a greater level of adaptability, ensuring its compatibility with a wide array of applications and systems. A more detailed exploration of this field will be provided in a dedicated section later in this chapter.

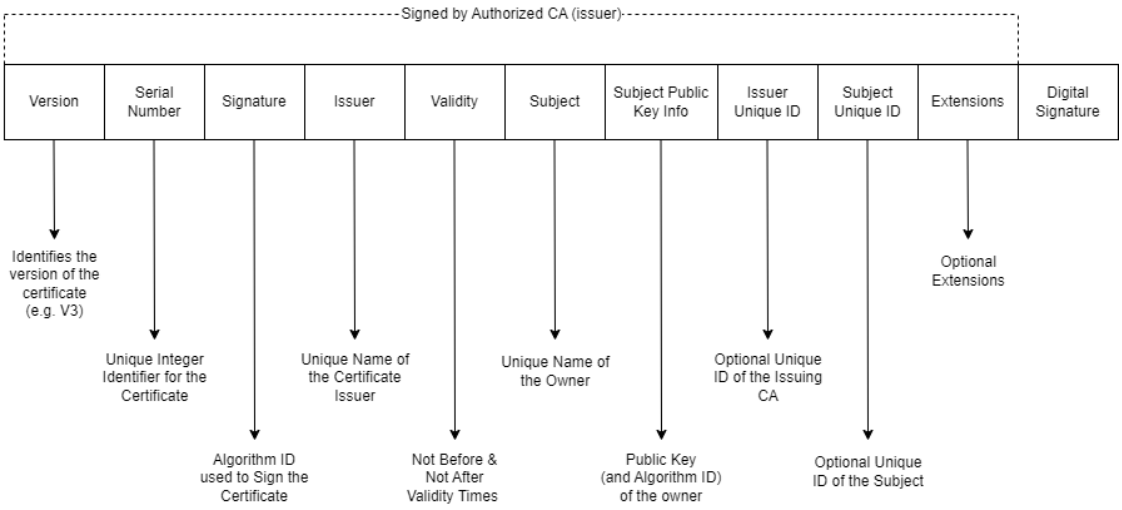

Figure 2.1: Structure of an X.509 certificate.

2.2.3 Certificate Revocation List

Inside a PKI-enabled system a certificate is issued but in case it is considered no longer trustable prior to its expiration date, it can be revoked; this can happen for instance if the private key is compromised, if the data inside the PKC is no longer accurate, or if a violation of the policies has occurred. Many environments support the temporarily block of a given certificate and, after some investigations, can become valid again.

X.509 standard defines another data structure called Certificate Revocation List (CRL), which is a periodic publication mechanism, signed by the Authority who issued the certificates, that contains a list of all the revoked and held certificates of the PKI. The structure of a CRL [8, Sec. 5] is the following (figure 2.2):

- *Version*: Version of the CRL. The latest is v2.
- *Signature*: OID of the algorithm used to compute the digital signature of the CRL.
- *Issuer*: unique DN of the CRL Issuer.
- *This Update*: The time that this CRL was issued, which may be represented in UTC.
- *Next Update*: The time by which the next CRL will be issued. Note that a new CRL may be issued before the Next Update time specified
- *Revoked Certificates*: The list of the revoked certificates that contains, for each entry, the unique Certificate Serial Number, the revocation Date/Time and optional per-Entry Extensions, that include:
  - *Reason Code*: the reason the certificate was revoked, some of them are `unspecified` (code `0`), `keyCompromise` (code `1`), `certificateHold` (code `6`), `privilegeWithdrawn` (code `9`).
  - *Certificate Issuer*: the name of the certificate issuer. Particularly useful for indirect CRLs.
  - *Hold Instruction Code*: this code is used to support temporary suspension of the certificates. They can be subsequently reinstated or permanently revoked.
  - *Invalidity date*: known or suspected time that the certificate was no longer considered valid.

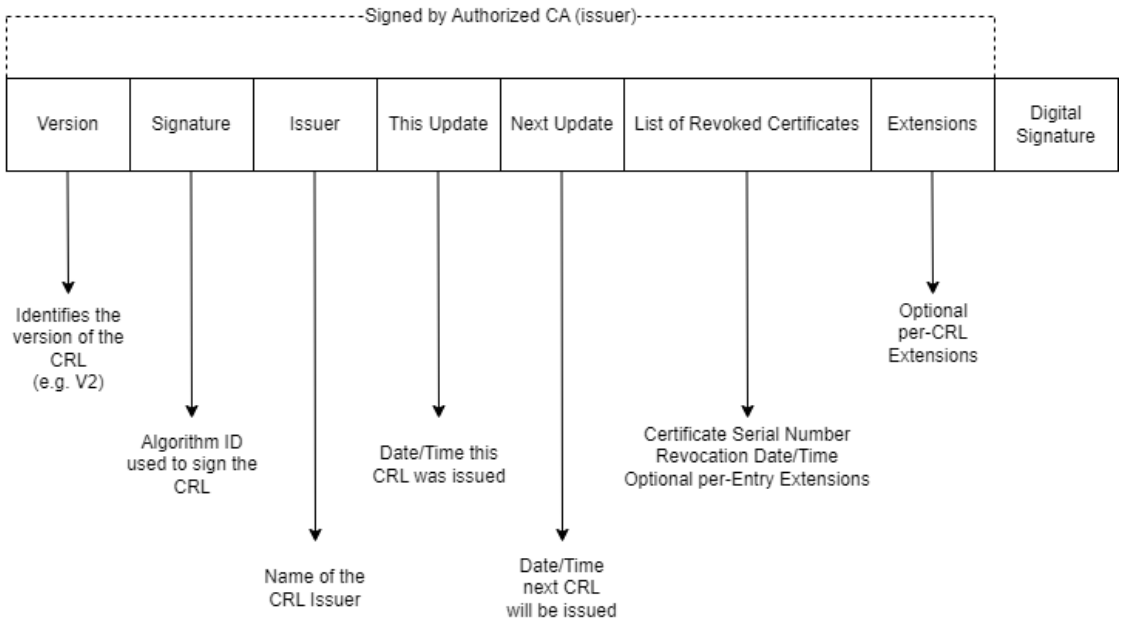

Figure 2.2: Structure of an X.509 CRL.

However, storing all the revoked certificate's info in a single file may be feasible in a small PKI domain, but when the number of EEs inside a community is big and constantly increasing, Complete CRLs can be difficult to manage due to two main problems. First of all, there will be an issue of scalability because the CRL publications will become voluminous over time; then, if the size of the CRL grows, it is reasonable to expect that its validity period increases as well, otherwise continual downloading of large files would lead to a bad network performance, but this can derive security issues because PKCs with recent revocations can still be used during the gap time between the old and the new CRL.

Here it comes the exigence of a smaller CRL, called Delta CRL, which consists in storing the information of revoked certificates during the period between the release of a base CRL and the next one. This approach allows whoever needs to validate a certificate to stay up to date about the revocation information of an Authority and without downloading large size files every time.

Another important concept that is useful to understand is the Indirect CRL (iCRL), which contain revocation information supplied from multiple CAs in a single file. Those are particularly useful when a system has many issuers, and rather than force a TTP to retrieve a CRL for each CA, all the data can be obtained reading the iCRL. Delta Indirect CRL is also supported.

2.2.4 Certificate Policies and Certification Practice Statement

A given certificate may have several policy-related extensions, which are particularly significant since they can govern the use of the certificate across various PKI domains, ensuring compliance with established policies. In RFC-3647 a Certificate Policy (CP) [9, Sec. 3.1] is defined as:

> a named set of rules that indicates the applicability of a certificate to a particular community and/or class of application with common security requirements.

Such policy is defined by a policy authority (e.g a government) that operates inside a policy domain (e.g. the region it administrates), hence all the certificates issued by the Certification Authorities of that domain must be compliant at least with the rules specified in the CP in order to be considered valid within that community. A Certification Practice Statement (CPS) [9, Sec. 3.4] is defined as:

> A statement of the practices which a certification authority employs in issuing certificates.

Basically it is a detailed, and often sensitive, document specific for a single CA where all the operations and procedures to meet the CPs requirements are described.

2.2.5 X.509v3 certificate Extensions

An X.509v3 certificate has two types of Extensions [8, Sec. 4.2]:

- *Public*: such extensions have already been defined in the standard, they have their own Object Identifier (OID), so anybody should be aware of their existence.
- *Private*: custom extensions, unique for a given PKI, that can be understood only by the entities belonging to that system.

Furthermore the extensions can be defined as *critical* or *non-critical*. During a secure transaction, when a certificate is presented, there is an entity, the Relying Party (RP), who verifies it. If at least one of the fields marked as critical are not recognized, the certificate must be rejected and the transaction fails; instead, if an extension marked as non-critical is unknown, the operation may proceed or it can be decided to decline the certificate anyway. Public extensions are divided in four classes [8, Sec. 4.2.1]:

1. *Key and policy information*
    - *Authority Key Identifier (AKI)*: unique identifier of the public key corresponding to the private key used to sign a certificate. This attribute is particularly useful when an issuer has multiple signing keys, because it helps to distinguish the correct keypair. Although this is always marked as non-critical extension, it must be included in the certificate to facilitate the certification path construction. The Identifier should be derived from the public key through the computation of a digest.
    - *Subject Key Identifier (SKI)*: Similar to the AKI, it identifies the public key of the subject. Also this extension must be non-critical and application are not required to verify it when performing the certification path validation.
    - *Key Usage (KU)*: a 9-bit string that defines the purpose of the key contained in the certificate (e.g. for encryption/decryption, signature etc.). This extension can be critical or non-critical, but, if present, the key must be employed for the scopes for which the corresponding option is defined.
    - *Extended Key Usage*: it can be used to provide additional information about the key or as substituted to the KU. The difference between the two extensions is that Key Usage specify the cryptographic operations of the key bound to the certificate, while this one is oriented towards applications. The fields are `serverAuth`, `clientAuth`, `codeSigning`, `emailProtection`, `timeStamping`, and `OCSPSigning`; in RFC-5280 for each of them it is recommended to associate one or more Key Usage bits to enhance consistency.
    - *Private key usage period*: it defines the validity period of the private key. It is not to confuse with the Validity attribute, which refers to the certificate only; in fact, in this case the certificate expires but the public key can be still associated to another certificate issued by the same, or different authority. Instead, after the Private key usage period the key-pair expires and cannot be used any more and a new one must be generated. This extension is always non-critical.
    - *Certificate Policies*: this extension can be critical or non-critical. It is a list of policies that indicate the terms under which the certificate can be used and it is written in form of a text message, URI or an OID. In case this extension is marked as critical, the processing application must adhere to at least one of the policies indicated, otherwise the certificate is not to be used.

- *Policy mappings*: it is present only in CA certificates as a non-critical extension, and indicates the correspondence of policies among different certification domains. For example if a CA may adhere to a given CP, and another one follows a different set of rules, a mapping in the certificate is useful for the analysis and comparison of the CPs of different CAs.
- *Inhibit anyPolicy*: this extension can be used in CAs certificates and must be marked as critical. It forbids the use of `anyPolicy`, which indicates that the certificate can be used for any purpose that is consistent with the other constraints in the certificate.

2. *Certificate subject and certificate issuer attributes*

   - *Subject Aternative Name (SAN)*: additional options to identify the Subject (certificate owner), that can include an e-mail, an IP or a MAC address, a URL. There can be more than one SAN and if the Subject attribute is left empty, this field must be critical.
   - *Issuer Alternative Name (IAN)*: similar to the SAN but for the CA that issued the certificate or CRL. It is not as important as the extension above.
   - *Subject Directory Attributes*: it contains a sequence of extra identification attributes of the subject. It must be non-critical.

3. *Certificate path constraints*

   - *Basic Constraints (BC)*: by the means of a flag, this extension identify whether the subject of the certificate is an End Entity (`cA = false`) or a CA (`cA = true`), and, in the last case, it is possible to specify the maximum depth of the certification sub-tree, the `pathLenConstraint`. For example if a certificate has this value set to 2, it means that that CA can certify another CA which in turn can certify another one only. It can be marked as critical or non-critical, but it is strongly suggested to always set to critical because otherwise an End Entity can act as a CA and issue other certificates.
   - *Name Constraints (NC)*: present in CA certificates only, it indicates the name restrictions of the subjects that can be certified. It only applies to the SAN and there are two fields, the *Permitted Subtrees* which is a whitelist of the required subtree names, and the *Excluded Subtrees*, a blacklist. It can be critical or non-critical.
   - *Policy Constraints (PC)*: present in CA certificates only, it can be used either to prohibit policy mapping or require that each certificate in a path contain an acceptable policy identifier. If the `inhibitPolicyMapping` field is present, the value indicates the number of additional certificates that may appear in the path before policy mapping is no longer permitted. If the `requireExplicitPolicy` field is present, it indicates the number of additional certificates that may appear in the path bedore an explicit policy is required for the entire path. According to RFC-5280 this field must be defined as critical.

4. *CRL Distribution Point*

   - *CRL Distribution Points*: this extension specifies where it is possible to download the CRL related to the certificate. Although this field can be critical or not critical, it is important for an application to know this information to check if the certificate is valid or not.
   - *Freshest CRL (or Delta CRL Distribution Point)*: it specifies where it is possible to download the Delta CRLs which is an update or supplement to the latest full CRL.

Private extension are created by the single PKIs for a closed group of entities, although the syntax stays the same, the semantics of such fields need to be defined. Their use is not recommended since they do not allow interoperability, but there are two important private extensions, the Private Internet Extensions defined by IETF, which are considered public since their use is common in the Internet user community [8, Sec. 4.2.2].

- *Subject Information Access (SIA)*: this extension was created to obtain more information beyond those specified in the Subject and SAN attributes; two fields are defined:

  - `accessMethod`: it indicates how additional information about the subject can be obtained. It supports protocols like HTTP and LDAP.
  - `accessLocation`: it provides a name or address, like an URL, where such information can be found.

  SIA is particularly useful in scenarios where a directory for certificate distribution is not employed, hence provides an alternative and more flexible way to obtain information about the certificate owner, such as a link to a web page, a phone number, or even an image.

- *Authority Information Access (AIA)*: a really important extension; it consists in a back pointer that goes from one issued certificate to the service offered by the CA. The most relevant field is `certStatus` that provides the address of the OCSP Server. It can be critical or non-critical, but it is recommended to set as non-critical. If the owner of the certificate is the CA itself, this field's name would be CA Information Access (CAIA), which fulfils the same function but is a self-pointer.

2.2.6 X.509 CRLv2 Extensions

As the PKCs, also the CRLs have their Extensions field [8, Sec. 5.2.1] and some of the concepts have already been analysed:

- *Authority Key Identifier*: similar to the AKI of the certificate, identifies the PK corresponding to the SK used to sign the CRL.
- *Issuer Alternative Name*: additional identity information associated to the issuer of the CRL.
- *CRL Number*: non-critical extension that represents a monotonically increasing sequence number, i.e. starting from CRL `#0`, the next one will be `#1` and so on. This is particularly useful for a user to determine when a particular CRL supersedes another one.
- *Delta CRL Indicator*: critical extension that indicates a CRL as being a delta CRL.
- *Issuing Distribution Point*: a pointer to a server to download the CRL.

2.3 PKI mechanisms

2.3.1 The Certification Architecture

The main actors of a PKI Framework are now presented:

- *Certification Authority (CA)*: in the previous sections it has been mentioned multiple times, it is the Trusted Third Party that issues and revokes PKCs and CRLs, and confirms their authenticity. There are two types of CA:
  - *Root CA*: also called Trusted Root, has a self-signed certificate that must be configured as a trusted root for all the clients inside the PKI. Verification of other certificates in the PKI ends with RootCAs self-signed certificate.
  - *Sub CA*: this CA is signed by another CA which can be the root or another subordinate. SubCAs certificate does not have to be configured as a trusted root because it is part of a certificate chain that ends in the RootCA
- *Registration Authority (RA)*: in many cases, to enhance scalability and decrease operational costs, it makes sense to delegate some of the CA functions to other entities, especially if the PKI system is really wide such as a big company. The RA is trusted to:
  - Establish and confirm the identity of an individual during the initialization process.

  - Provide end users with shared secrets for future authentication in an online initialization process.
  - Begin the certification process with a CA for individual users, which includes registering specific attributes linked to the user.
  - Generate keying material for end users.
  - Handle various key/certificate lifecycle management tasks, like starting a revocation request or key recovery for an end entity.
  - It is never allowed to issue certificates or CRLs.

  There can be more than one RA connected to a single CA.

- *Validation Authority (VA)*: trusted by the CA, provides information on whether a certificate is currently valid or not. Its task is to distribute the CRLs of the Authorities for which it operates, or to check PKCs validity through the Online Certificate Status Protocol (OCSP), which will discussed later.

2.3.2 Certification generation

The process of issuing a certificate can be performed in several ways. A solution could be that an entity generates its own key-pair, then saves its SK locally in a protected format and send the PK along with the associated attributes to the CA as a Certificate Signing Request (CSR). The certificate is not issued immediately because the RA must check the validity of the information given by the requester, examining its identifier. Depending on the policy, such identifier may vary and can be for instance an ID card, a fingerprint or just name and surname. Then the RA sends the data to the CA and, if the outcome of the request is good, the PKC is generated and returned to the supplicant. All the issued certificates are distributed in a public repository which contains also the CRLs.

Another method could be that the RA generates the key-pair, obtains the PKC and distributes them on a secure device.

2.3.3 Trust Models

In the previous sections the word "trust" has been mentioned many times. It is a really important and complex concept that represents the basis of a PKI-enabled system; without trust among the entities all the notions stated above would decay. ITU-T gives this definition [10, Sec. 3.3.52]:

> Generally, an entity can be said to "trust" a second entity when it (the first entity) makes the assumption that the second entity will behave exactly as the first entity expects. This trust may apply only for some specific function. The key role of trust in this framework is to describe the relationship between an authenticating entity and a authority; an entity shall be certain that it can trust the authority to create only valid and reliable certificates.

Hence, trust deals with assumptions, expectations, and behaviour, and there are risks associated with it. For instance if the Trusted Authority is compromised, vulnerable or there has been a misplaced reliance, all the certificates are breached as well. A CA builds its trust in several ways, e.g. if it is certified by a more established and widely recognised Authority, or it has raised a good reputation over time, or for its transparency and compliance with standards and strict policies.

In a PKI, several trust models exist here are some examples:

- *Strict Hierarchy of CAs*: there is an important and reliable Root-CA that signs other CAs, which, in turn, sign other CAs that can issue the certificates to End Entities. The result is a "chain of trust" where it is easy to track and verify the certificate path from the end entity to the root.

- *Distributed Trust Architecture*: it is built when two root-CAs trust each other by issuing a cross-certificate (a certificate where a issued by a Root-CA for another Root-CA), or by means of a third CA (a bridge CA) cross-certified by each Root-CA. In this case the trust path is more difficult to follow for applications.

2.4 Standards and Protocols

In order to enhance the security and the functionalities of a PKI, many protocols and standards have been defined over time. In this section the main frameworks and guidelines will be analysed, some of them will be applied during this project.

2.4.1 TLS/SSL

Even if they are not specific of the PKI, Transport Layer Security and Secure Sockets Layers are closely related to the framework as they use digital certificates issued by CAs to authenticate the identity of parties involved in the communication, such as a web browser and a server. The protocol ensures that data transmitted between the entities remains encrypted and secure.

In detail, when a user visits a website that uses TLS, the web server sends its TLS certificate to the user's web browser, which checks the validity of the certificate. If this is trusted, the browser establishes a secure connection (HTTPS) with the server. While clients usually don't require a TLS certificate, they may be necessary for authentication in enterprise environments, where clients have to prove their identity before accessing sensitive resources, or in mutual TLS, where both server and clients must trust each other.

SSL is the predecessor of TLS, but the two terms are often interchangeably used, despite TLS being the more secure and updated version. PKI's role in the ecosystem of TLS/SSL consists in managing, distributing, and revoking these digital certificates, ensuring the trustworthiness of secure internet connections.

2.4.2 OCSP

Online Certificate Status Protocol, described in RFC-6960 [11], consists in a client requesting a server, namely the VA, if a given certificate is valid at the moment of the request. To avoid fake responses, those are digitally signed by the server, and the possible answers can be:

- Good
- Revoked, giving also revocationTime and revocationReason.
- Unknown, if the certificate does not belong to the PKI domain of the validating server.

There are several methods for OCSP to get information about revocations, for instance it can simply read the public CRL repositories, fetch the data directly from the CA database or asking other OCSP servers like a chain. The responses are managed by OCSP responders, which can be operated directly by the CA itself or a trusted or delegated responder.

One big limitation of OCSP is that it only provides real time verification, so it is useful for immediate transactions, but delayed operations are not supported by this protocol.

2.4.3 PKCS

Public Key Cryptography Standards are a set of 15 guidelines conceived and published by RSA Security LLC starting from the early '90s, some of them are deprecated but others serve as a benchmark for today's IT systems. The most relevant are:

- *PKCS#1 – RSA Cryptography Standard*: RFC-8017. It states the mathematical properties and format of RSA public and private keys and the basic algorithms and encoding/padding schemes for performing RSA encryption, decryption, and producing and verifying signatures.
- *PKCS#3 – Diffie-Hellman Key Agreement Standard*: the protocol that allows two parties to jointly establish a shared secret key over an insecure channel.
- *PKCS#7 – Cryptographic Message Syntax Standard*: RFC-2315. It is used to sign and/or encrypt messages under a PKI and for certificate transmission.
- *PKCS#8 – Private-Key Information Syntax Standard*: RFC-5958. It is used to store private key with or without encryption.
- *PKCS#10 – Certification Request Standard*: RFC-2986. The format of messages sent to a CA to request certification of a public key (CSR), the response will be possibly a PKCS#7 certificate.
- *PKCS#11 – Cryptographic Token Interface*: an API defining a generic interface to cryptographic tokens. Chapter 4 focuses entirely on configuring and managing PKCS#11.
- *PKCS#12 – Personal Information Exchange Syntax Standard*: RFC-7292. It defines a container format commonly used to store private keys along with PKCs, protected with a password-based symmetric key.

2.4.4 SCEP

Simple Certificate Enrollment Protocol, introduced in RFC-8894, was designed by CISCO for the secure issuance of certificates over non-secure networks and automates the process of certificate distribution for devices and applications, supporting operations like enrolment, renewal, and revocation of certificates [12]. This protocol has the following characteristics [13]:

- Uses a request/response model over HTTP (GET method and optionally POST).
- Exclusively supports RSA-based cryptography.
- Employs PKCS#10 for certificate requests formatting and PKCS#7 for signed/encrypted message conveyance.
- Supports asynchronous certificate granting by the server, requiring regular polling by the requester to check the status of the certificate issuance.
- Provides limited support for CRL retrieval.
- Online certificate revocation is not supported.
- The enrolment process requires a pre-shared challenge password, to be written in a field within the CSR.

SCEP is being superseded by EST protocol.

2.4.5 EST

Like SCEP, Enrollment over Secure Transport, RFC-7030, simplifies the issuance of certificates but using HTTPS for secure transport and TLS for client and server security, hence it is inherently more secure than the predecessor [14]. EST addresses several limitations of earlier protocols by incorporating modern cryptographic standards and streamlining certificate management processes. Its main features are:

- Utilises HTTPS, leveraging TLS to authenticate the client and the server and to protect communications between the two parties. This enhances the trust model in certificate issuance and ensures that all the data transferred during the certificate processes is encrypted and secure from interception.

- Supports Elliptic Curve Cryptography (ECC), which is known for its strength and efficiency in generating secure cryptographic keys. This modern approach allows for faster computations and reduced resource consumption, making it suitable for both high-volume environments and devices with limited computational power.

- Provides mechanisms for automated certificate renewals, which simplifies the management of certificate lifecycles by reducing the manual overhead required in processes like re-enrollment.

- Ensures that both the client and the server authenticate each other with TLS mechanisms, enhancing the overall trust model in certificate issuance.

EST is progressively becoming the preferred protocol for secure certificate management.

2.4.6 CMP

RFC-4210 explains the Certificate Management Protocol, designed to facilitate various certificate-related operations in a secure and automated manner. CMP is the oldest among the enrolment protocols, its principal characteristics and operational aspects are now elucidated: [15]:

- It supports a wide array of operations crucial for comprehensive lifecycle management of certificates. This includes initial enrolment, periodic renewal, immediate revocation in case of compromise, and recovery of lost or corrupted certificates.

- The employed security mechanisms to provide integrity and confidentiality include the use of digital signatures to verify the authenticity of communication, encryption to safeguard data in transit, and detailed authorization checks to ensure that only legitimate requests are processed. This multi-layered security approach ensures that CMP can operate securely over various network environments, including less secure ones.

- Unlike many other protocols that require specific transport layers, CMP can be utilized over any transport protocol.

- It uses the Certificate Request Message Format (CRMF), described in RFC-4211, which allows for a more nuanced and detailed specification of certificate and key attributes than simpler formats such as PKCS#10. CRMF supports features such as specification of multiple public keys, setting preferences for certificate template attributes, and providing detailed instructions to the CA on how certificates should be handled.

- This protocol supports both asynchronous and synchronous modes of operation, providing flexibility in how responses are received. In asynchronous mode, requests can be polled at intervals, while responses are immediate in synchronous mode.

- CMP is designed to handle errors robustly, providing comprehensive feedback on the nature of any issues encountered during the certificate management process. This includes detailed error messages and status codes that help diagnose and resolve issues efficiently, enhancing the reliability of the protocol.

Due to its broader range of functionalities and flexibility, CMP is a really complex protocol.

2.4.7 ACME

Automated Certificate Management Environment, RFC-8555, is a protocol designed by the Internet Security Research Group (ISRG) initially for its own project, Let's Encrypt, an open CA that provides domain validated certificate for free. Today many PKIs and browsers support ACME [16].

ACME allows organizations to automate processes such as CSR generation, domain ownership verification, certificate issuance, and installation. It is primarily used for obtaining domain validated certificates, as they only require verification of domain existence. Higher-value certificates like organization validated and extended validation can also be obtained using ACME, but additional support mechanisms are necessary. The objective of this protocol is to set up an HTTPS server and automate the provisioning of trusted certificates and eliminate manual transactions [17]. The main features of this protocol are:

- It automates the verification of domain ownership, which is essential for issuing domain-validated certificates. This process typically involves the CA challenging the applicant to prove control over a domain, either via HTTP, DNS, or TLS.
- While primarily used for domain-validated certificates, ACME also supports obtaining higher-level certificates such as organization validated and extended validation certificates. These, however, require additional validation mechanisms that go beyond simple domain control verification.
- The protocol automates not just the issuance but also the renewal and revocation of certificates, facilitating ongoing management without manual intervention.
- It is designed to be efficient and minimize the number of interactions required to obtain a certificate, thereby speeding up the process and reducing the load on both the client and the server.
- It offers several methods for proving control of a domain, giving clients flexibility in how they fulfil CA challenges based on their specific network and security configurations.

ACME is becoming the protocol of choice for automated management of TLS certificates, particularly due to its support from major browsers and its integral role in the widespread adoption of HTTPS across the web.

2.4.8 Comparing CMP, SCEP, EST and ACME

The four protocols analysed before are basically designed to achieve the same purpose, i.e. simplify and automatise the enrolment process of a PKC, along with the main stages of the certificate's lifecycle (renewal, revocation and recovery). Each one of has its own advantages and drawbacks, which make them more or less suitable under given circumstances [18].

- CMP is the oldest of the protocols. It is one of the most encompassing protocols in terms of available operations and various permutations, making it really versatile and flexible. Nevertheless, this strength is also its main weakness because it makes the framework really complex and rather client-unfriendly. Its flexibility make it ideal for environments that require detailed and granular control over certificate operations, such as large corporations or telecom industries.
- SCEP is nearly as old as CMP and was designed to be as lightweight and simple as possible, in contrast with the other protocol. Its range of operations is limited, but still supports enrolling and retrieving certificates and CRLs. The user-friendly interface is the main advantage, but the disadvantages are the limited cryptographic agility, since it supports only RSA keys, and the messages are transferred over insecure channels (HTTP). This protocol continues to be useful in certain scenarios, particularly within legacy systems, where updating infrastructure to support newer protocols might be challenging.

- EST is developed as an improvement over SCEP, it is simple and lightweight, relies on HTTPS and allows support for Elliptic Curve Cryptography. The design incorporates a number of security features, including server-side key generation, which adds an additional layer of security by preventing the exposure of keys. However, its functionality is somewhat constrained in comparison to more comprehensive protocols such as CMP, limiting its use within environments that prioritise security but require less complex certificate management operations.
- ACME is the best protocol for managing TLS certificates in web environments. It allows clients to verify domain ownership through challenges, enabling auto-renewal and broad TLS adoption. Despite its strengths in automation and extensibility for TLS, ACME's reliance on specific client-server interactions and challenge processing may limit its applicability outside standard TLS use cases, requiring both server and client support for any protocol extensions or variations.

2.5 PKI in practice

To conclude, PKI is a really useful and effective authentication technology that identifies entities in an environment by means of public key cryptography, along with all the mechanisms described above. It is important to note that the identification of an individual does not inherently provide insights into their capabilities, establish automatic trust, or guarantee possession of all necessary information for executing specific transaction steps. Many transactions necessitate data regarding an entity's authorization, entitlement, or privileges. PKI, while crucial in confirming an entity's identity, does not directly assist in these aspects. It primarily serves to protect the integrity, authenticity, and confidentiality of such information, which originates and is utilized in systems outside of PKI. In synthesis, PKI:

- does not provide authorization (although it may be used to protect authorization information).
- does not create trust in other entities (although, beginning with a trust model, it can lead to trust that a given key is associated with a given name).
- does not name entities (even if there was no PKI, I would still need to be able to decide if this e-mail is from a "Fred" I know or a "Fred" I don't know).
- does not make things secure (even if it may be used as a basis for some aspects of security, all other security-related system/network tools and techniques still need to be utilized).
- does not correct bad human behaviour (people still need to be trained not to do "bad" things; malicious users–especially malicious administrators–can still exist).

A Public Key Infrastructure provides authentication – no more, no less. [7, Chap. 14] In the next chapters all the introduced concepts will be utilized to design a PKI-enabled system with the open-source framework EJBCA CE.

Chapter 3

EJBCA Framework

3.1 EJBCA Overview

The EJBCA (Enterprise JavaBeans Certificate Authority) Framework is a Public Key Infrastructure software solution that provides CA services [19]. It is designed to offer a scalable, robust, and customizable platform for managing digital certificates and PKI ecosystems. The application is developed by PrimeKey Solutions AB, a Swedish company engaged in PKI services, which is now part of the American firm Keyfactor, and it is widely used by governments, financial institutions, and other organizations to manage digital certificates and provide secure communication across networks.

There are two versions of EJBCA, Community Edition (CE), that offers basic PKI functionality and is ideal for testing and evaluation; and Enterprise Edition (EE), designed for larger organizations and government agencies, that supports more complex needs with additional security features and compliance guarantees not available in the free release [20]. CE will be utilized in this study due to its open-source nature, in order to align with the objective of this Master's Degree Thesis project, which is analysing the behaviour of a PKI environment using open-source tools, as introduced in Chapter 1.

3.1.1 EJBCA Concepts

The main EJBCA concepts correspond to the core components of a typical PKI already introduced in Chapter 2, and in addition there are several framework-specific elements that help to manage the authorities and the supported features [21]:

- *Certification Authority (CA)*: the application supports the creation of Root CAs and Sub CAs.
- *Registration Authority (RA)*
- *Validation Authority (VA)*
- *End Entity (EE)*: it is the user of the PKI and can be a device, person or server and it is always located at the end point of a hierarchy of certificates. The EE requests a certficate from the CA or RA and can hold many certificates that will have all the same identifying values.
- *Profiles*: templates used to define rules and constraints for different classes of entities or certificates that have one or more attributes in common. They are particularly useful because they are customizable and ease the process of issuing certificate. There are two types of *profiles*:

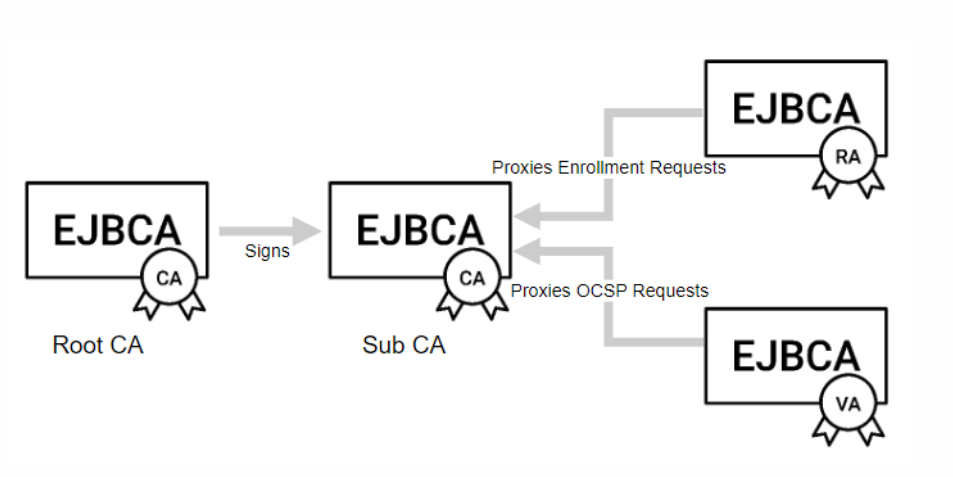

Figure 3.1: Example of PKI Hierarchy in EJBCA (source: PrimeKey).

  - *End Entity Profile*: it allows for the customization of enrolment constraints. Although there is a default "Empty" profile with no restrictions, it is good practice to set specific limitations on registration values. This kind of *profile* dictate direct EE attributes and certificate identifying fields, excluding key and signature data, which are determined by Certificate Profiles. Values within these profiles range from fixed, modifiable, mandatory, to optional, enabling both specificity and flexibility across individual or multiple end entities.
  - *Certificate Profile*: it defines the specific attributes and constraints of certificates, including extensions, algorithms, and key sizes, essentially outlining the limitations of an issued certificate. This kind of *profile* is designed to be generic and shared across different End Entity Profiles, allowing for versatility in certificate issuance. Extensions within a Certificate Profile can be set as present and critical, with some having fixed values and others being user- or certificate-specific. Certificate Profiles are very important for CA configuration because they safely separate the creation of CA's and EE's certificates.

- *Crypto Token*: in HSMs the crypto token is the part of a slot which contains that slot's cryptographic keys. In EJBCA a Crypto token is a set of cryptographic keys which may be protected by a password and can be activated or deactivated. The platform supports two types of Crypto Tokens:

  - *Soft Crypto Tokens* which are stored in the database as PKCS#12 files protected by a password.
  - *PKCS#11 Crypto Tokens* which are stored inside a HSM and are accessed through the PKCS#11 API.

  Note that each CA has one and only one Crypto Token, though several CAs could share the same Crypto Token.

- *Publisher*: it stores issued certificates to a central location, e.g. a publisher can periodically send CRLs to an Active Directory or an SCP Server to allow authorized VAs and other entities to safely access these data.

- *Internal Key Binding*: it links a key pair within a Crypto Token to non-CA uses in EJBCA, acting as a simplified, purpose-driven key store. It references a specific key pair, includes a non-CA certificate, and may list trusted certificates for verifying trusted sources. For instance, an OcspKeyBinding utilizes a key from a HSM, accessed via a Crypto Token, to sign OCSP responses for a CA, with properties to define the OCSP response's validity period. This mechanism enhances flexibility and security in key management.

3.1.2 Supported algorithms

EJBCA supports many cryptographic algorithms such as DSA 1024, RSA (from 1024 to 8192), ECDSA (different curve types) and also quantum-safe algorithms (FALCON and DILITHIUM). The last ones would be the optimum in terms of security, but due to hardware limitations, ECDSA and RSA are considered valid alternatives for this project. In particular, ECDSA Prime Curves will be adopted as singing key because it enhances scalability and efficiency, providing the same level of security of RSA while using much shorter key lengths, for example an ECDSA prime256 key is equivalent to a RSA 3072 key [22], [23]. Prime curves are preferred to other types like Gost, Brainpool or Binary, because they have a good balance between security and performance [24].

On the other hand, RSA is a more mature algorithm and it is widely supported across other platforms and systems, so choosing it for the encryption keys would be better in order to establish secure communications channels, and enhance compatibility and standards compliance.

3.1.3 Keystores

When the certificates are generated, it is possible to download them in the form of different container formats, called *keystores*, EJBCA supports the following ones:

- *DER*: this format has extension .der and encodes the ASN.1 syntax of a certificate in binary.
- *PEM*: this format is the base64-encoded version of the DER file, it is more commonly used than the previous one because it can be easily read using a text editor with a base64 decoder. This container format may include just the public certificate, just the private key or an entire certificate chain including public key, root certificates and/or private key. Usually a PEM certificate can be found with extensions `.pem`, `.crt`, `.cert`, `.crt` and, if it only contains a private key, `.key`. Since the text of this kind of file is in clear, it is strongly recommended to protect those containing the secret key using encryption.
- *PKCS#12*: a password protected container format that include both private key and public certificate. This keystore is widely supported, the content is fully encrypted and it is the best method to store and transport the keypairs securely. It can be found with extensions `.pkcs12`, `.pfx`, `.p12`.
- *PKCS#7*: does not contain the private key and it is used for certificate interchange, the extension are `.p7b`, `p7c` and `.keystore`.
- *JKS*: Java Key Store format, `.jks`, supported by Java-based server to store private and/or public keys. This format is in disuse in favour of PKCS#12.
- *BCFKS*: a keystore format from the Bouncy Castle Java library, designed to meet FIPS (Federal Information Processing Standards) requirements. It has extension `.bcfks` and can store both private and public keys and is used in environments requiring compliance with FIPS.

The choice of the proper keystore depends on the application's requirements, interoperability needs and security policies, but anyway a format can be easily converted into another one using suitable tools like OpenSSL or Java Keytool. In this project, PKCS#12 and PEM will be chosen as standard keystores.

3.2 First set-up

In this section all the steps to create a new instance of EJBCA Community Edition are shown [25]. EJBCA was installed in a containerized environment using Docker and Docker Compose on a Virtual Machine running the CentOS 7 Linux distribution. First of all, two separate bridged networks were defined, *access-bridge* is used by EJBCA instance to be reachable from the outside,

the other one *application-bridge* is used to allow communication between EJBCA container and MariaDB.

The port mapping between host and container is specified as follows:

- `8080` of the EJBCA container is mapped to port `80` of the host to allow HTTP communication (`8080` is the port exposed by *WildFly*, the open-source application server on which EJBCA relies on)
- `8443` of the EJBCA container is mapped to port `443` of the host to allow HTTPS communication

In Docker Compose, if the host IP address is not explicitly specified in the port mapping, it is assumed to be `0.0.0.0`, i.e. all the interfaces. As a result, the web page would be reachable not only by the localhost interface, but also from the LAN. During configuration phase, it is recommended to set `127.0.0.1:80` as port mapping so that the EJBCA instance can be reached only by the local machine. Note that in both the network configurations, the web pages can be reached through `https://0.0.0.0/ejbca/...` or `https://localhost/ejbca/...` (or `http://`) because the browser will redirect `0.0.0.0` to `localhost`, but this does not mean that the service is available in the local network. The YAML file used in the Docker Compose configuration is shown in Code 3.1.

```yaml
networks:
  access-bridge:
    driver: bridge
  application-bridge:
    driver: bridge
services:
  ejbca-database:
    container_name: ejbca-database
    image: "library/mariadb:latest"
    networks:
      - application-bridge
    environment:
      - MYSQL_ROOT_PASSWORD=foo123
      - MYSQL_DATABASE=ejbca
      - MYSQL_USER=ejbca
      - MYSQL_PASSWORD=ejbca
    volumes:
      - ./datadbdir:/var/lib/mysql:rw
  ejbca-node1:
    hostname: ejbca-node1
    container_name: ejbca
    image: keyfactor/ejbca-ce:latest
    depends_on:
      - ejbca-database
    networks:
      - access-bridge
      - application-bridge
    environment:
      - DATABASE_JDBC_URL=jdbc:mariadb://ejbca-database:3306/ejbca?characterEncoding=UTF-8
      - LOG_LEVEL_APP=INFO
      - LOG_LEVEL_SERVER=INFO
      - TLS_SETUP_ENABLED=simple
    ports:
      - "80:8080"
      - "443:8443"
```

Code 3.1: Configuration file for Docker Compose to start an EJBCA CE instance (source: EJBCA Documentation).

During the first startup, the application automatically creates a default self signed CA called *ManagementCA*, that issues the first TLS certificate for the server which will be used to establish a HTTPS connection with the clients.

To become familiar with the main functions of the framework, the first step was to set up a *stand-alone PKI architecture* (Fig. 3.2), where all the authorities and database were configured together on a single machine. This is the simplest configuration, but the best security practice would be to separate the CAs from RAs and VAs into different servers to isolate the CA from the external internet as much as possible. Other possible configurations will be discussed later, after understanding the core concepts of the framework.

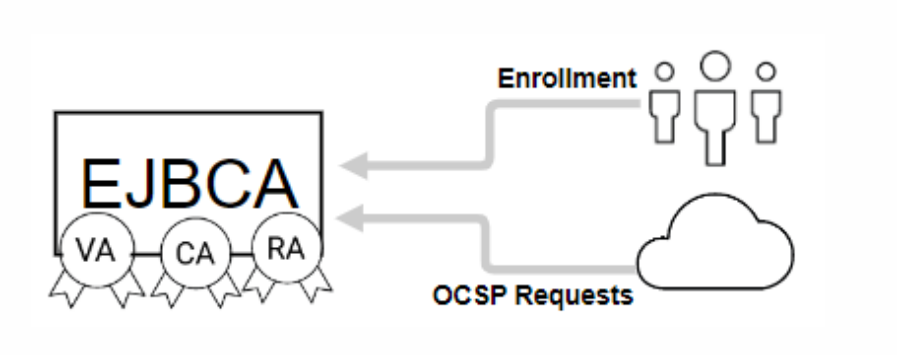

Figure 3.2: Stand-alone CA/RA/VA (source: PrimeKey).

3.2.1 Issue a SuperAdmin certificate

Once EJBCA is correctly installed in the container, the interface can be accessed at `localhost/ejbca/adminweb` for the administration page, both in HTTP (port 80) and HTTPS protocols (port 443). Before version 8.0 there was a public webpage in `localhost/ejbca` that served as RA, but it has been deprecated now and replaced with the page `localhost/ejbca/ra`, which is more configurable in terms of permissions.

While trying to access the administration page without HTTPS, a "Permission denied" message will be displayed even though no form of client authentication has yet been configured, instead, accessing via HTTPS, an interface like Figure 3.3) will appear.

At this point, it is strongly recommended to restrict access to the admin page by configuring HTTPS client authentication and, to achieve this, the first task requires ManagementCA to issue a X.509 certificate for the administrator through the public RA web page (Fig. 3.4).

The certificate could have the following settings:

- Certificate subtype: ENDUSER;
- Key-pair generation: By the CA. Although this practice is discouraged, an exception is made in this case since the certificate is being issued to the administrator;
- Key algorithm: RSA 3072 bits;
- Common Name: SuperAdmin;
- Key Recoverable: No. For security reason, the key will not be stored inside the CA;
- Username (`superadmin`) and Enrollment code (i.e. a password);

Then, the digital certificate can be downloaded as PKCS#12, JKS or BCFKS formats. The chosen one will be PKCS#12 because the browser (Firefox or Chromium) supports this standard.

To proceed to the next step, it is important to understand what permissions, roles and members are and how they work in EJBCA. Permissions control what users can do, roles group permissions

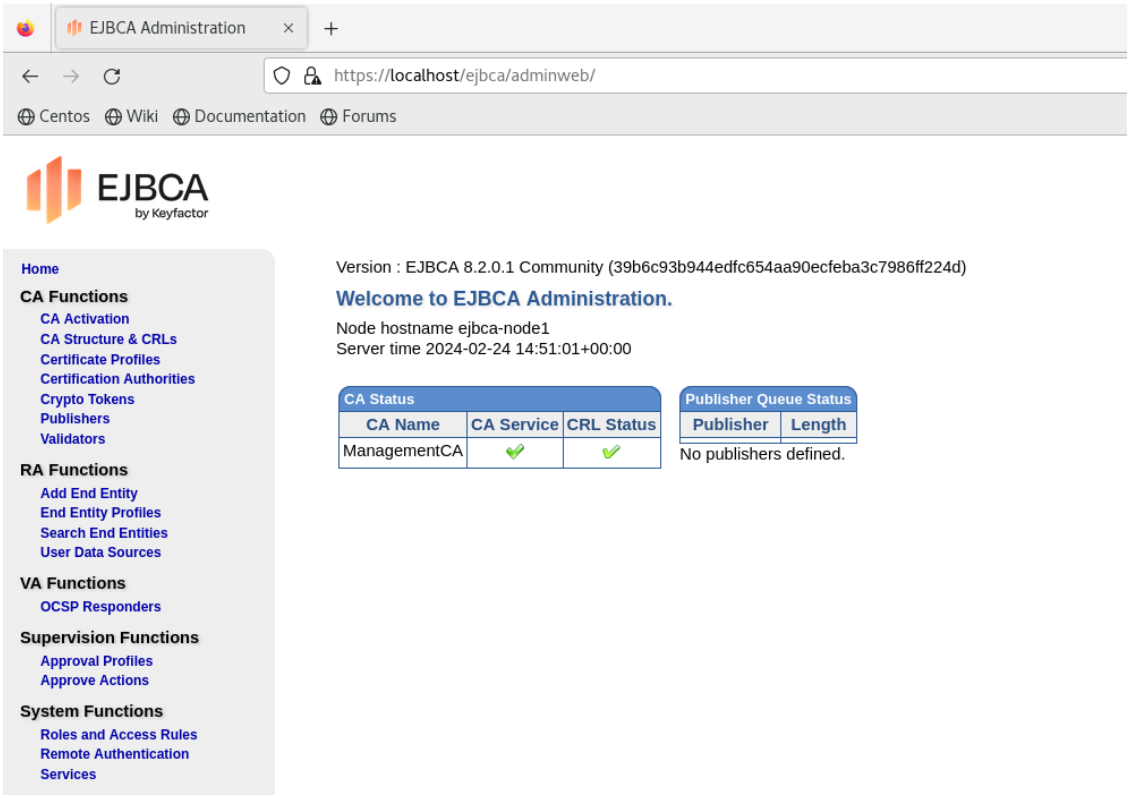

Figure 3.3: Default EJBCA Admin Page with the default CA ManagementCA.

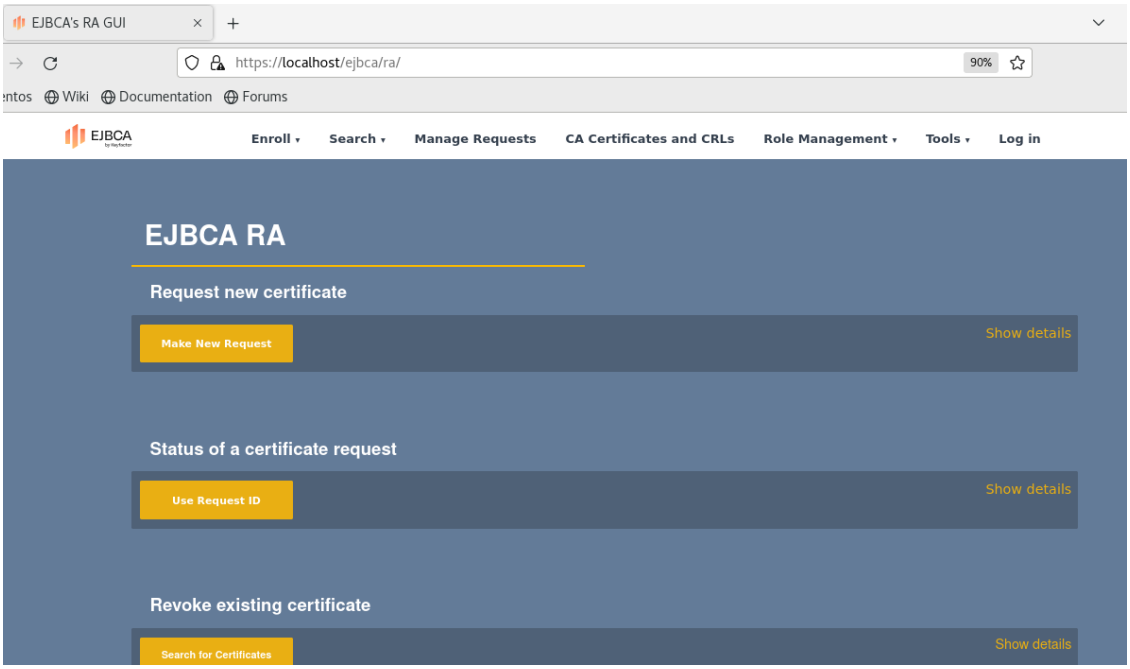

Figure 3.4: Default EJBCA RA Page.

together, and members are users or group of users who are assigned roles. Users' level of access is based on the permissions associated with their roles. This system makes it easy to control access to functions and resources in the Certificate Authority and will be better analysed later.

In order to make the previously created user an administrator of the PKI, in the Admin Web it is necessary to navigate through the Roles and Access Rules page (Fig. 3.5). Here, it is suggested

to limit the accesses to the public by deleting the "Public Access Role". This role can be recreated in a second moment, when the infrastructure is ready to use, to manage accesses and permissions for the Public Access (RA webpage).

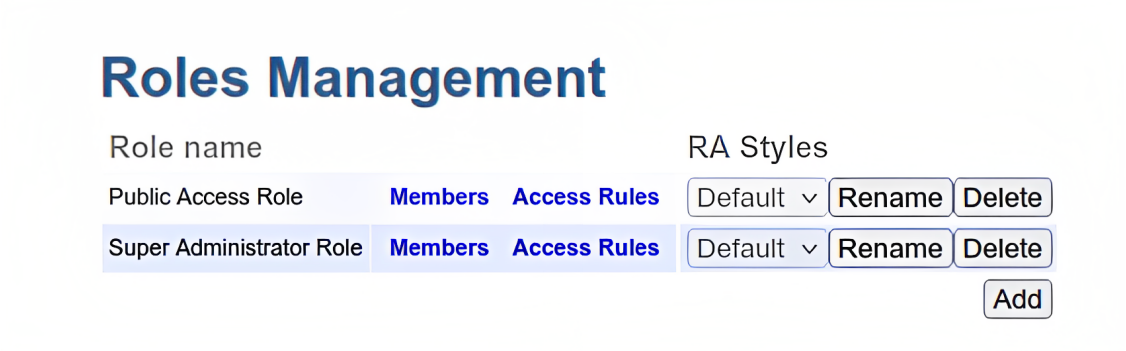

Figure 3.5: EJBCA default roles.

Then, in the "Members" section of "Super Administrator Role" (Fig. 3.6), specify the following fields for the admin role:

- Match with: Select "X509: CN, Common name".
- CA: Select "ManagementCA" for the CA to match on.
- Match Value: The CN value from the created certificate: "SuperAdmin". Note that this is a case-sensitive matching.

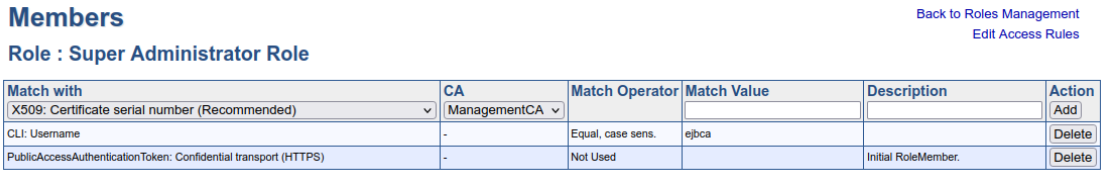

Figure 3.6: Default Super Administrator Role members.

After restarting the Docker to save the changes, the "PublicAccessAuthenticationToken" should be removed from the "Super Administrator" role list to finally limit access (Fig. 3.7). This ensures that only the holder of the digital certificate with the Common Name (CN) "SuperAdmin", issued by this specific "ManagementCA", has exclusive access rights to the EJBCA page as an administrator.

Now, every time that users try to visit `https://localhost/ejbca/adminweb`, they will be requested by the browser to show the valid SuperAdmin certificate otherwise the server will forbid the access (Fig. 3.8).

One last thing to mention is that SuperAdmin is not the only entity that serves as Super Administrator. In fact, there is another entry in the list which is not a certificate, but the user created by the container whose username is "ejbca". This is important because it is exploited by WildFly to start up the application and it allows a human operator to control and debug the framework directly through the Command Line Interface, hence it cannot be deleted. Anyway it is important to protect the machine that hosts EJBCA and limit the accesses to the console because otherwise the infrastructure can be seriously compromised.

3.2.2 Create a PKI hierarchy

The next task is creating a new CA and its Sub CA to better understand all the settings parameters inside EJBCA environment.

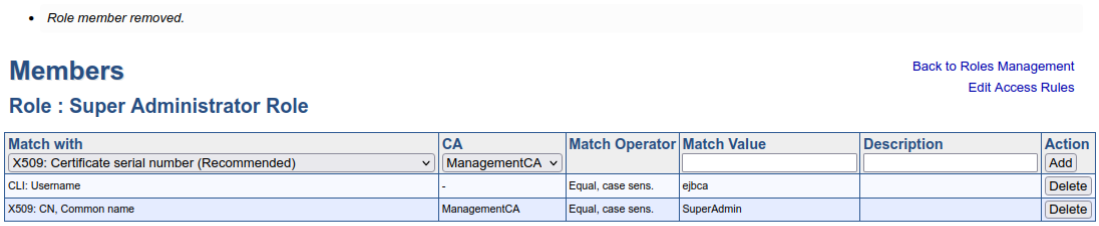

Figure 3.7: Super Administrator Role members after changes.

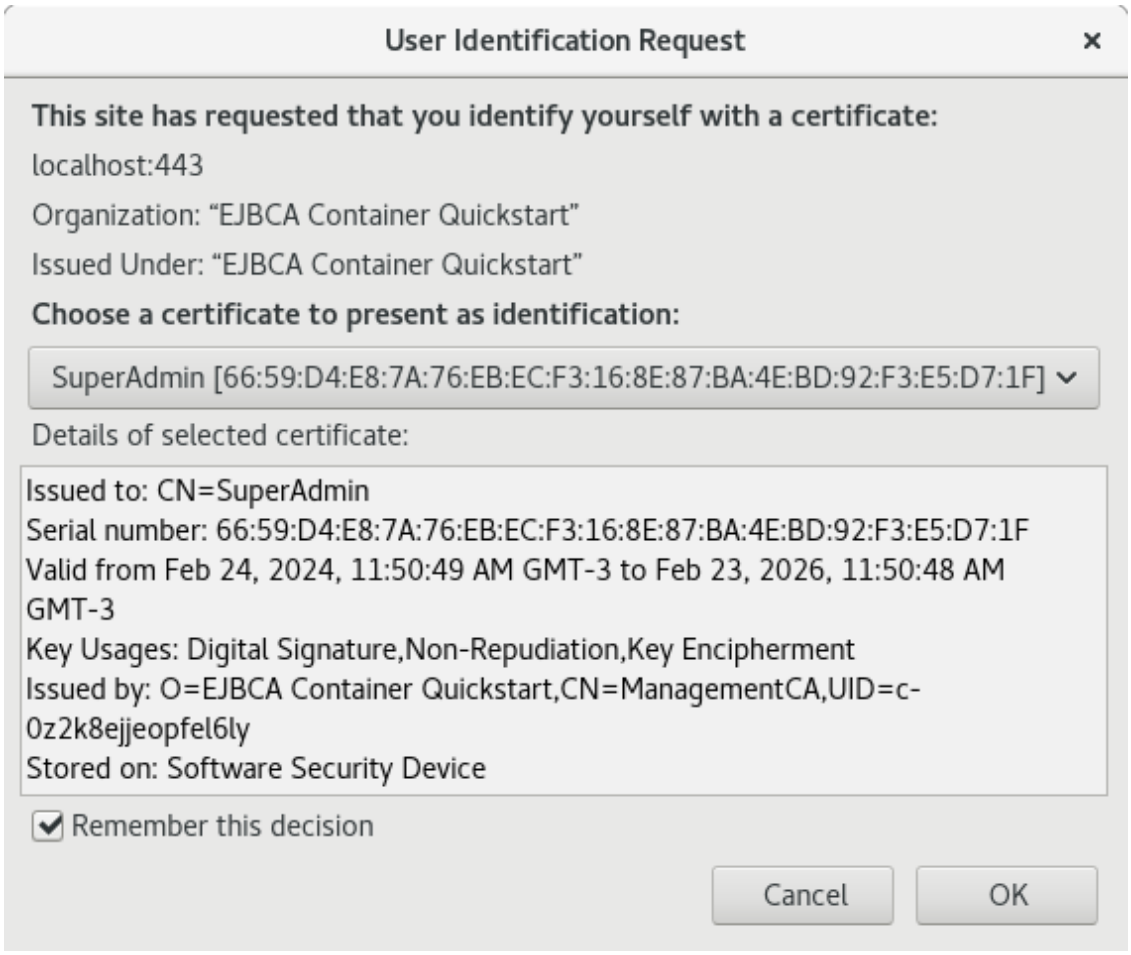

Figure 3.8: Identification window to access the Admin Web Page.

From the framework it is possible to create a new Authority starting from a predefined profile that can be edited so that it accomplishes with the company's policy. There are plenty of options that can be set inside the menu of a CA profile, the main are:

- *Key Algorithms*: cryptographic algorithms available for use when issuing certificates with that profile. The choice should be made based on the specific needs and requirements of the organization, balancing security and performance considerations.
- *Validity of the certificate*: how long the certificate should last. It can be set an expiration date or its duration, i.e. year, months, days until seconds. Generally speaking a digital certificate for a CA can stay active from 5 to 10 years, it is important to balance the validity period because a shorter time reduce the impact of any potential compromise or vulnerability in the certificate, while longer periods can reduce costs and operational overhead.

- *Override permissions*: this options allow the user to set some settings outside the limits set by the Certificate Profile, such as validity period, extensions, basic constraints etc.
- *x.509v3 extensions*: additional fields that can be added to digital certificates to provide additional information or functionality and can be used by clients to make decisions about whether to trust the certificate or not.
- *Qualified Certificate Statements extension*: indicate that a certificate holder is in compliance with certain security standards, such as those specified by government or industry regulations.

After setting the profiles both for the RootCA and its SubCA, it is necessary to create two Crypto Tokens, each containing 3 keypairs:

- *Signing key*: the primary key pair, used to sign digital certificates, CRLs and OCSP responses.
- *Encryption key*: used for encrypting key recovery information (only if the key recoverable option was enabled during key pair generation when issuing a certificate).
- *Test key*: used to sign test certificates or CRLs and it is not trusted by default. Using a test key allows the CA to test new certificate profiles, policies, and workflows without impacting the security and integrity of the production keys. This ensures that any issues or errors can be identified and resolved before they impact the security of the PKI system.

Once the keys and the two profiles are created, it is time to create the two CAs. A Crypto Token would be assigned to the Root CA, which would self sign its digital certificate and would have, for example, "Polito RootCA" as Common Name (CN), "Politecnico di Torino" as Organization Name (O) and "IT" as Country Identifier (C). The other one would belong to the Sub CA, whose certificate would be signed by the "Polito RootCA" and would have "Polito SubCA" as CN, and same O and C of the Root. Note that this is just an example of a PKI hierarchy, a Sub CA could have a different O and/or C.

Having a Sub CA in the PKI Infrastructure is really important because it enhances scalability, flexibility and provides an additional level of security. However, a PKI infrastructure for small organizations, that has a limited number of certificates or limited use cases, a single Certification Authority may be sufficient.

3.2.3 Issue TLS Certificates

After setting up the CA, digital certificates for End Entities can be issued. This section outlines the steps for interacting with the Registration Authority and request either a server or client TLS certificate.

Before creating a TLS server profile, the CA must create a specific Certificate Profile in the CA functions menu and then an End Entity Profile from the RA menu. When creating the EE Profile, it is important to consider how a server can be recognized, such as by its Common Name (i.e. its primary DNS), Organization, Country, and its DNS (which could be more than one), then it is necessary to set the Extended Key Usage for "Server Authentication". Finally it is necessary to bind this profile to the previously created TLS Certificate Profile, determine which CA can issue this type of certificate. For security reasons, it is preferable to create a user-generated key.

Once the profile is created, a server or the RA on its behalf can request its certificate from the RA Webpage. It is possible to let the CA generate the keypair but for security reasons, it is preferable to create a user-generated key. The entity will upload the Certificate Signing Request (CSR) that can be easily generated using a tool such as OpenSSL, which contains all the information about the applicant and its public key. After entering the username and perhaps other optional information (such as the DNS), the digital certificate can be downloaded.

The same steps can be performed to create a TLS Client Certificate, with the difference that this Certificate Profile has "Client Authentication" as Extended Key Usage, and other information, such as the client's birthdate and/or email, may be requested instead of the DNS.

Figure 3.9 and Figure 3.10 report two examples of TLS Certificates. However, it is important to note that while this is a basic configuration, digital certificates should have many X.509 attributes because they provide additional information that can be used to establish trust in the identity of the certificate holder.

View Certificates

Username	**polito_tls**
Certificate number	1 of 1
Certificate Type/Version	X.509 v.3
Certificate Serial Number	70AA98390D18AE06198932A99BD3D29FF2B8EC39
Issuer DN	CN=PoliTO Sub CA,O=Politecnico di Torino,C=IT
Valid from	2024-03-05 09:54:05+00:00
Valid to	2025-03-05 09:54:04+00:00
Subject DN	**CN=www.polito.it,O=Politecnico di Torino,C=IT**
Subject Alternative Name	
Subject Directory Attributes	None
Public key	EC (prime256v1): 113B6C2F841B4FF8E70B59841E608EAEF351068A6C5EEEC...
Basic constraints	**End Entity**
Key usage	Digital Signature
Extended key usage	Server Authentication
Name constraints	No
Authority Information Access	OCSP Service Locator URI: http://ejbca-ca-node:80/ejbca/publicweb/status/ocsp
Qualified Certificates Statements	No
Certificate Transparency SCTs	No
Signature Algorithm	SHA512WITHECDSA
Fingerprint SHA-256	E14E9CA18636BCC2044F78ADA69FEBF9 116677C3AFBA5092B9E794BAB6B48A69
Fingerprint SHA-1	E0127403B872B0C5B6A053360DC3978E0CA44E9B
Revoked	No

Figure 3.9: Example of TLS Server Certificate.

3.3 Advanced configurations

After a first look on the main functionalities of EJBCA, it is time to explore further in detail advanced supported settings that enhance the security and the efficiency of the Public Key Infrastructure.

3.3.1 Services

The framework supports various services that run on timely basis, performing different background tasks. Among the available ones, here are described the most relevant services that will be eventually used for this project:

View Certificates

Username	**s292486**
Certificate number	1 of 3
< View Older	
Certificate Type/Version	X.509 v.3
Certificate Serial Number	7BD72345DBE1153339E93E277273698F9771F65B
Issuer DN	CN=PoliTO Sub CA,O=Politecnico di Torino,C=IT
Valid from	2024-03-05 09:57:37+00:00
Valid to	2025-03-05 09:57:36+00:00
Subject DN	**CN=Alessandro Loconsolo,O=Politecnico di Torino,C=IT**
Subject Alternative Name	
Subject Directory Attributes	countryOfCitizenship=IT, gender=M, dateOfBirth=19980904
Public key	EC (prime256v1): E94248DFC125D5FB30396C0C679C7B5B781348874E3A499...
Basic constraints	**End Entity**
Key usage	Digital Signature
Extended key usage	Client Authentication
Name constraints	No
Authority Information Access	OCSP Service Locator URI: http://ejbca-ca-node:80/ejbca/publicweb/status/ocsp
Qualified Certificates Statements	No
Certificate Transparency SCTs	No
Signature Algorithm	SHA512WITHECDSA
Fingerprint SHA-256	5EF47C83E17E4312BF82783817B42BF1 45F18BCFA0A1F63A9171D080B202E8A2
Fingerprint SHA-1	23B9DFDC989D2A599882EEA063E72D3283E0DF61
Revoked	No

Figure 3.10: Example of TLS Client Certificate.

- *Certificate and CRL Reader*: populates a VA with certificates and CRLs sent from a CA via an SCP Publisher.
- *Certificate Expiration Check*: checks if a CA has certificates about to expire and sends a notification email to the end user and/or the administrator.
- *CRL Download and Update*: another way to populate the VA, periodically downloads a CRL form the provided URL (distribution point) and updates the revocation information for the certificates.
- *CRL Updater*: checks if any of the configured CAs need a new CRL and generates it if necessary.
- *HSM Keepalive*: this service will periodically test all available crypto tokens by performing a test signing in order to avoid session timeouts to the HSM.
- *Renew CA*: if a CA's certificate is about to expire this procedure automatically renews it.

Each service is composed of a worker, which is the class that will be executed when the service runs; an interval determining the period of activity of the service; and an optional action, or actions, that should be executed under specific circumstances.

3.3.2 Page permissions

The framework allows to customise the authorizations with a good degree of freedom, in fact, it is possible to create roles with different permissions levels and bind certificates to them. For instance, if SuperAdmin is completely free to manage all the aspects of the infrastructure, there are subordinate roles that can be created in order to restrict the access only to the fields of interest of given entities. This function enhances scalability and security, and it is really important because provides support for the least privilege principle, i.e. each entity must be able to access only the information and resources that are necessary for its legitimate purpose.

There are several built-in role templates, that ease the configuration using default values, the available roles are:

- *CA Administrators*: they have full access to the CA and RA functions of the CA (or CAs) they administrate. CAs and related end entities and certificates will be hidden if the administrator does not have access.
- *RA Administrators*: they have access to the Enrollment, Search and Manage Requests pages, depending on the selected End Entity Rules. Access is restricted according to the selected CAs and end entity profiles as well.
- *Supervisors*: they have access to the Manage Requests and Search pages only, in read-only mode.
- *Auditors*: they have access to everything in read-only mode, except for the Enrollment page which is not accessible.

It is also possible to manually configure rules and permissions by creating customised roles. For instance, EJBCA can be configured for public access by setting a Public Access Role that allows users and systems outside the organization to securely request certificates and perform key management operations in the RA web.

3.3.3 Certificate issuance

The EJBCA framework offers several ways to issue a certificate, and the choice is up to the designer, depending on the needs of the PKI.

A first method is registering the End Entity navigating to the “Add End Entity” option, located in the sidebar menu among the RA Functions in the Admin Web page. Here the RA (or the authorised Authority) can start the enrolment process selecting the proper EE Profile, the issuing CA, and then entering all the requested data. It is possible to decide whether the keypair will be user generated or created by the CA (Fig. 3.11. After creating the End Entity, the certificate is not issued yet and the Authority will give username and enrolment code to the entity that will conclude the registration process. To do so, the entity needs to go to the RA Web and enrol via username (in the horizontal menu bar “Enroll”, then “Use Username” using the credentials that have been given during the registration phase. Finally, if the keypair generation has been left to the CA, all the user needs to do is to download the PKCS#12 certificate, otherwise using a tool like OpenSSL, has to generate a private key and a CSR to send to the RA and then download the certificate in PEM format, PKCS#7 or others (only public part). It is suggested to let the end user generate the keypair because this reduces the risk of key compromise during transmission or storage by third parties, however this operation requires a more advantaged knowledge by the user.

A second configuration can be directly letting the entity to proceed with the Request by itself, via the RA Web page, navigating to “Make New Request” from the homepage. Here, as seen in the previous section, it is possible to select which type of certificate is going to be requested among the available ones, the issuing CA and possibly deciding how to generate the key (depending on the administrator configurations). In case of using User Generated, the CSR must be sent. At this point it is necessary to wait for the CA to accept the request, the system will generate a Request

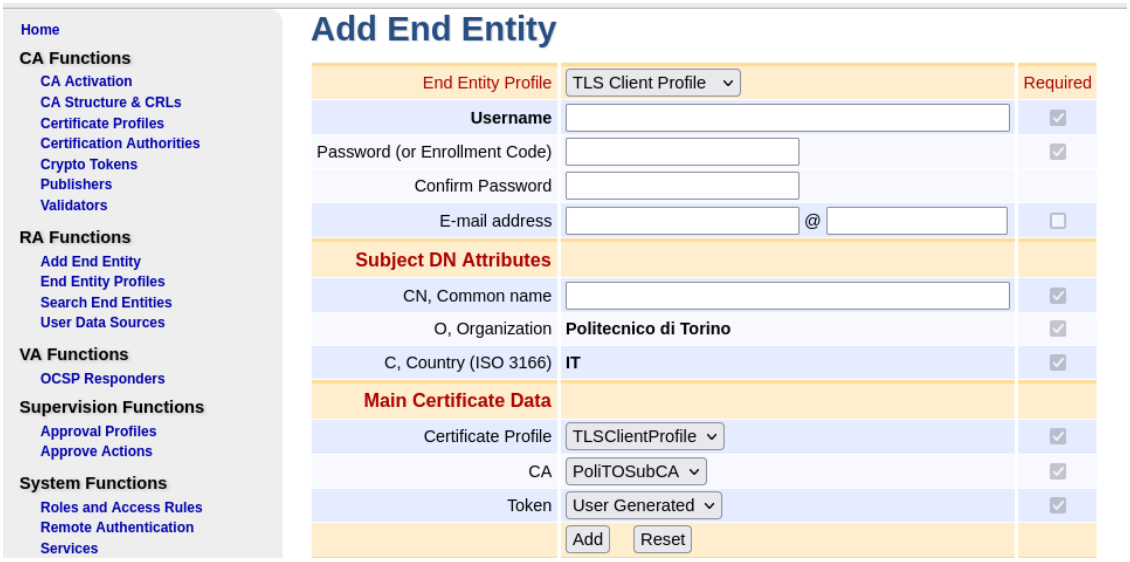

Figure 3.11: Add End Entity UI.

ID that can be used to check the status of the request. After the approval it is possible to finalise the process and download the certificate. Under specific needs the PKI designer can also decide to remove the step of the Request ID and directly let the end user to enrol with automatic approval by the CA, by the way this method is not recommended.

In order to create a Certificate Signing Request with OpenSSL it is necessary to create a configuration file like Fig. 3.12 that usually has .cnf as extension.

```
1 [ req ]
2 default_md = sha256
3 prompt = no
4 distinguished_name = dn
5
6 [ dn ]
7 CN = s123456
8 O = Politecnico di Torino
9 C = IT
```

Figure 3.12: Example of configuration file for the CSR.

Usually the request sample is given by the RA and the End Entity will only complete it with its own data. Then, along with the .cnf file, it is necessary to create a private key executing the OpenSSL command:

```
openssl genpkey -algorithm <algorithm> -pkeyopt <option>:<value> -out
    <output_file.key>
```

Where `algorithm` is the public key algorithm to use (e.g. `rsa`, `ec`, etc.) and the option `pkeyopt` is used to specify the type of the elliptic curve, the key length etc. (e.g. `rsa_keygen_bits:4096`, `ec_paramgen_curve:prime256v1`, etc.). Then, to create the CSR it is necessary to type the following command:

```
openssl req -new -key <private_key.key> -config <request_file.cnf>
```

After that, the tool will return a message encoded in base-64 which contains the configuration informations, the public key derived from the private component and the digital signature. The CSR has to be sent to the RA (Fig. 3.13) and eventually it is possible to continue the enrolment process.

```
---BEGIN CERTIFICATE REQUEST-----
MIIBADCBpwIBADBFMRYWFAYDVQQDDA13d3cucG9saXRvLml0MR4wHAYDVQQKDBVQ
b2xpdGVjbmljbyBkaSBUb3Jpbm8xCzAJBgNVBAYTAklUMFkwEwYHKoZIzj0CAQYI
KoZIzj0DAQcDQgAEETtsL4QbT/jnC1mEHmCOrvNRBopsXu7CxdjlSNjkXjEyasbc
JhvCrhGobxW5IIJvAYpv5dzZMbSE2PjIocDzLqAAMAoGCCgGSM49BAMCA0gÁMEUC
IGpWp5uMVZjhZM+kntZN5qr56kcaVLDE84HMRNYzBY7sAiEA5G6Y7sm9GEst4nI6
u3eTrN5P3Iblg3QSD0s06fzsbbA=
---END CERTIFICATE REQUEST-----
```

Figure 3.13: Example of CSR that is about to be submitted to the RA.

3.3.4 High Availability

In a PKI ecosystem redundancy is key in order to ensure that the failure of a single instance does not result in downtime, this capability is also called high availability. While redundancy for VAs and RAs can be achieved using multiple independent instances, for CAs this can be done thanks to database clustering, i.e. multiple independent nodes accessing the same (highly available) database.

A possible and simple configuration can be having two datacenters, located in two different sites, and no load balancer (Fig 3.14), this will eventually provide a single layer of redundancy.

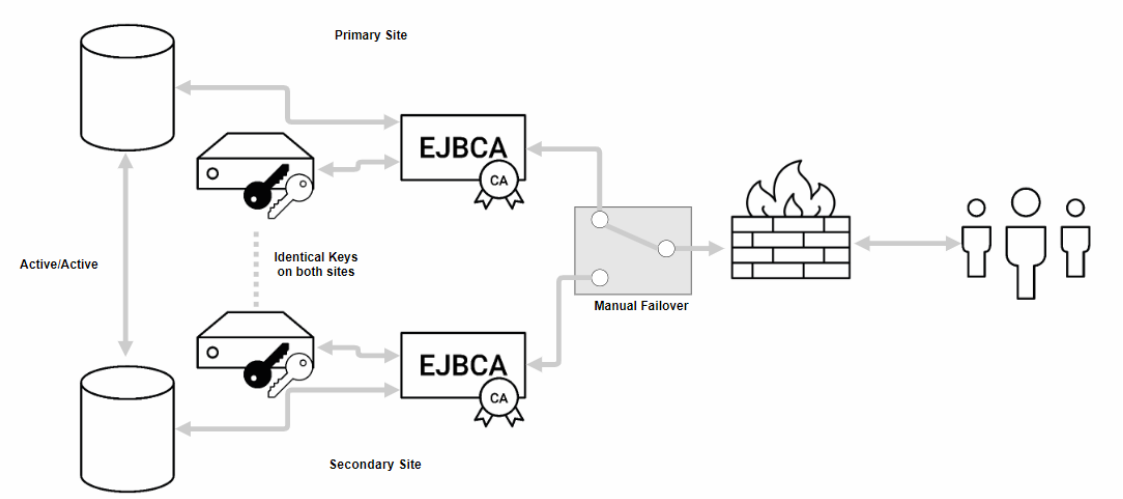

Figure 3.14: A Disaster Recovery Site with manual or automatic failover (source: Primekey).

In this design all the traffic is lead to the primary site during the normal operation, while the disaster recovery site is online but idle. If the primary server fails, the traffic will be directed to the secondary one with a manual or an automatic failover. The two databases are connected and configured as one primary and one replica, i.e. the disaster recovery is a copy and the synchronization should be synchronous to prevent data loss during the failure.

A more flexible, but more advanced, solution would be using a load balancer between multiple sites that all are active at the same time. This will produce multiple layers of redundancy and performance benefits.

3.4 Setting a VA in EJBCA CE

A robust PKI operates having the Authorities installed in separated instances with the CA placed behind a firewall Fig. 3.15. The Certification Authority, in fact, has to be protected by any means

because if compromised, the whole architecture would collapse; and that is why a configuration of this kind, that limits direct access to the CA to authorized entities only, is considered secure.

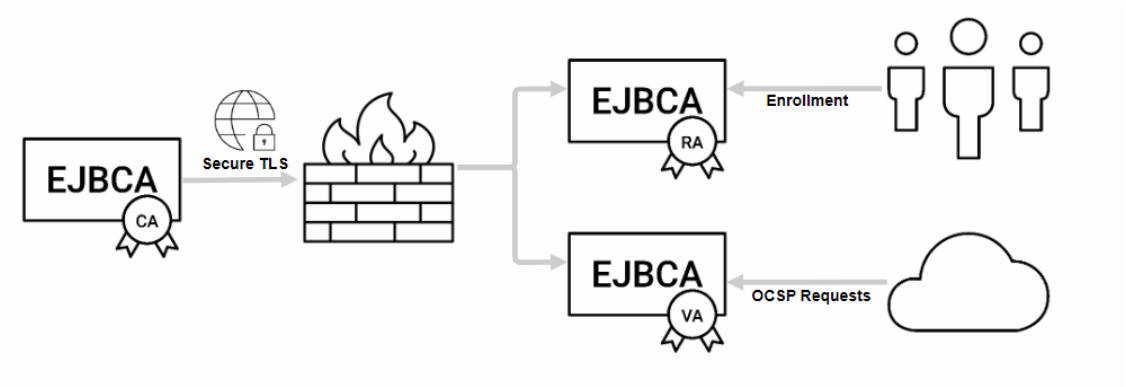

Figure 3.15: CA with distributed RAs and/or VAs (source: PrimeKey).

One strong limitation of EJBCA CE is that this configuration is not supported, this means that the CA must be exposed to the external and insecure network in order to issue certificates for the End Entities. In detail, the authorities that cannot be separated are the CA and the RA, but the VA can operate independently on a different instance. As a result, to limit the risk of CA compromise it is crucial to define the roles and the access permissions for each actor in the PKI; plus, having a distributed VA can benefit performance and security because allows the traffic to convey towards the CA/RA group in case of certificate issuance, renewal, retrieval or towards the VA in case of OCSP and validation requests.

The main challenge in configuring an external VA lies in defining how it will parse information on certificates issued by the CA it serves. The simplest way to achieve this, and the only one possible in EJBCA CE, is the periodical download of the latest CRLs issued by that CA. In this section all the steps to create a stand-alone VA using EJBCA CE will be explained and analysed.

3.4.1 Import CA truststores

The proposed architecture features both the CA/RA block and the VA block, each with its own and independent databases; the two modules are then connected with a network bridge that allows them to communicate with each other. The following Code 3.2 shows an example of the design.

```
networks:
  access-bridge:
    driver: bridge
  application-bridge-ca:
    driver: bridge
  application-bridge-va:
    driver: bridge
services:
  ejbca-database-ca:
    container_name: ejbca-database-ca
    image: "library/mariadb:latest"
    networks:
      - application-bridge-ca
    environment:
      - MYSQL_ROOT_PASSWORD=foo123
      - MYSQL_DATABASE=ejbca
      - MYSQL_USER=ejbca
      - MYSQL_PASSWORD=ejbca
    volumes:
      - ./datadbdir-ca:/var/lib/mysql:rw
  ejbca-database-va:
```

```
    container_name: ejbca-database-va
    image: "library/mariadb:latest"
    networks:
      - application-bridge-va
    environment:
      - MYSQL_ROOT_PASSWORD=foo123
      - MYSQL_DATABASE=ejbca
      - MYSQL_USER=ejbca
      - MYSQL_PASSWORD=ejbca
    volumes:
      - ./datadbdir-va:/var/lib/mysql:rw
  ejbca-ca-node:
    hostname: ejbca-ca-node
    container_name: ejbca-ca
    image: keyfactor/ejbca-ce:latest
    depends_on:
      - ejbca-database-ca
    networks:
      - application-bridge-ca
      - access-bridge
    environment:
      - DATABASE_JDBC_URL=jdbc:mariadb://ejbca-database-ca:3306/ejbca?characterEncoding=UTF-8
      - LOG_LEVEL_APP=INFO
      - LOG_LEVEL_SERVER=INFO
      - TLS_SETUP_ENABLED=simple
    volumes:
      - ./ca/truststore:/mnt/external/secrets/tls/ts
    ports:
      - "80:8080"
      - "443:8443"
  ejbca-va-node:
    hostname: ejbca-va-node
    container_name: ejbca-va
    image: keyfactor/ejbca-ce:latest
    depends_on:
      - ejbca-database-va
    networks:
      - application-bridge-va
      - access-bridge
    environment:
      - DATABASE_JDBC_URL=jdbc:mariadb://ejbca-database-va:3306/ejbca?characterEncoding=UTF-8
      - LOG_LEVEL_APP=INFO
      - LOG_LEVEL_SERVER=INFO
      - TLS_SETUP_ENABLED=simple
    volumes:
      - ./va/truststore:/mnt/external/secrets/tls/ts
    ports:
      - "81:8080"
      - "444:8443"
```

Code 3.2: Docker Compose configuration to set up an external Validation Authority.

Basically what is done here is a duplication of two EJBCA instances, each one has its own database connected with a dedicated network, “application-bridge-ca” and “application-bridge-va”, and then “access-bridge” links the two units. Actually, the databases are not duplicated, “ejbca-database-ca” is populated with the previously created CAs and End Entities, while “ejbca-database-va” is initialised from scratch.

The first thing to do is to export the truststore from “ejbca-ca-node” to “ejbca-va-node”. The truststore is a JKS file that contain all the public key of the trusted Authorities; in this case it is important that the VA knows the certificates to trust since the database is empty initially. WildFly server automatically creates a truststore if it is not detected in the container directory /

`mnt/external/secrets/tls/ts` and knowing this, so the following strategy is applied:

1. Run only "ejbca-ca-node" and its database executing the following command in the directory of the configuration file:

   ```
   docker compose up ejbca-database-ca ejbca-ca-node -d
   ```

2. Wait the server to be ready and access the "ejbca-ca" CLI container as root:

   ```
   docker exec -it -u root ejbca-ca /bin/bash
   ```

3. Find the `truststore.jks`, which is the file automatically created by the server at the startup, running the command:

   ```
   find / -type f -name "truststore.jks"
   ```

4. Copy the file into the directory `/mnt/external/secrets/tls/ts`, which is the mounted volume. The file is now visible from the host in `./ca/truststore` (starting from the directory of the configuration file).

5. Copy the truststore into `./va/truststore`.

6. Finally, run the VA and its database and the server will deploy the given truststore:

   ```
   docker compose up ejbca-database-va ejbca-va-node -d
   ```

Now, EJBCA VA WebPage is accessible from the host via `https://localhost:444/ejbca/adminweb`, specifying port `444` is important because in the default HTTPS port `443` stands the CA server. Here, it is necessary to limit the access, as done in the first set-up, and to use that "SuperAdmin" certificate created in Section 3.2.1 it is necessary to import the ManagementCA certificate (the "ManagementCA" that can be observed in the Web Page is created by the server during the startup and does NOT correspond to the "ManagementCA" created in the previous EJBCA instance). After setting the Super Administrator role properly, it is possible to proceed with the importation of the certificate of the CA that will sign the OCSP Responses.

3.4.2 Import CA certificate and download the CRLs

Before importing the CA certificate, it is necessary to edit the CA from the "Certification Authorities" page and set a proper Default CRL Distribution Point and the OCSP service Default URI, under "Default CA defined validation data" field (Fig. 3.16). Here, let the framework generate the link for the distribution point and if necessary adjust the port according to the configuration file. In this example, the URL will be `http://ejbca-ca-node:8080/ejbca/publicweb/webdist/certdist?cmd=crl&issuer=CN%3DPoliTO+Sub+CA%2CO%3DPolitecnico+di+Torino%2CC%3DIT`, whereas host-side it is seen as `http://localhost:80/ejbca/publicweb/...`

As OCSP service Default URI, instead, the link will be `http://ejbca-va-node:8080/ejbca/publicweb/status/ocsp`, i.e. `localhost:81/ejbca/publicweb/...`. This will be useful in the near future and specifies that the OCSP requests are to be queried to the VA node that is going to be configured.

After these introductory steps, it is possible to switch to the VA and import the CA certificate (the full chain) from the "Certification Authorities" page and, after that, edit the entry and add the previously generated CRL Distribution Point. Before proceeding it would be better to make sure the two containers are actually connected using the command `docker network inspect access-bridge` and the command `ping` from one end to the other.

If there are no connection issues, it is time to upload the CRL into the node, an operation that can be performed manually or automatising the process using CRL Downloader Service. Since CRLs are issued periodically, the second option is the way to go.

To add this service, it is necessary to navigate through the "Services" entry, from sidebar menu of the VA Admin Web page, under "System Functions"; then, give a name the service, add it to the list and edit it. In the "Edit Service" UI, it is possible to:

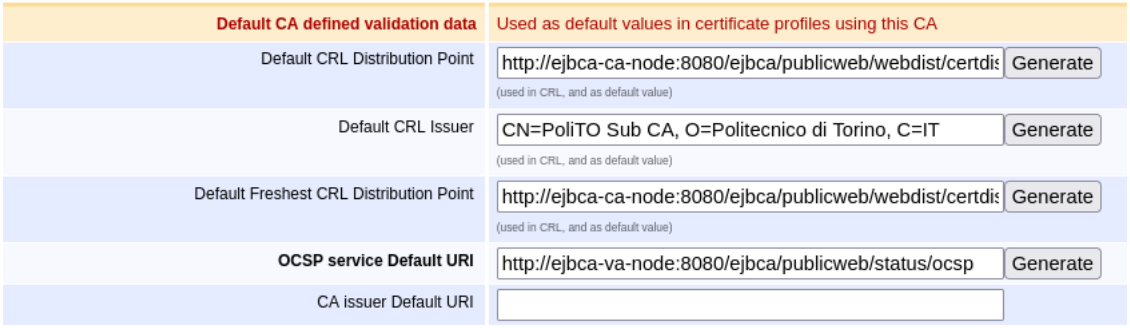

Figure 3.16: "Default CA defined validation data" field in the Edit CA page.

- Choose the worker, which in this case is "CRL Downloader".
- Select the CA(s) to check.
- Check or uncheck the flag "Ignore nextUpdate and always download the CRL", which, if checked, it will force the download of the CRL whenever the service is executed but if the last downloaded CRL does not coincide, if unchecked, instead, the service will only download the CRL when the last known CRL indicates that a new one is be available. For this configuration it is suggested to activate the option.
- Set the maximum allowed size to download, i.e. if the CRL exceeds this limit, the service will refuse to process it.
- Specify the activation period, i.e. how often the service wakes up and tries the download.
- Check or uncheck the "Active" flag, to turn the service on or off.

After the service activation, the latest CRL should be available in the "CA Structure and CRLs" page. The next stage consists in creating an OCSP Responder.

3.4.3 Set up an OCSP Responder

The phase of populating the VA database with the CRLs is over, now it is time to create a trusted OCSP Responder for the CA. To do so, in the VA it is necessary to create a new Crypto Token for the responder, along with a RSA keypair (ECC is not supported). After that, from the "OCSP Responders" page it is necessary to create, a new responder and bind the keypair to it, choose the signature algorithm, select the CAs it will generate responses for and, among all the properties, check "non existing is good". It is important to check this property because the VA does not keep track of the active certificates, but it only knows information about the revoked ones, hence a certificate issued by the CA which is not recognised by the VA is actually good because is not present in the list of the revoked certificates.

Then, to activate the Responder and make it a trusted one, it is necessary to bind a certificate signed by the CA. For this purpose, the framework allows to download an automatically generated CSR to send to the CA and request an OCSP Signer Certificate. To issue this kind of certificate it is necessary to go to the CA, create a Certificate Profile for the OCSP Signer, and create an End Entity profile to associate with that Certificate Profile. A certificate, in order to be specific for signing OCSP responses, must have Digital Signature only as Key Usage, and OCSP Signer as Extended Key Usage.

Now in the RA Web it is possible to proceed with the certificate enrolment using the CSR and issue the certificate, that finally can be uploaded in the VA (Fig. 3.17).

After this procedure the VA is ready to make signed OCSP responses on behalf of its CA.

Certificate Type/Version	X.509 v.3
Certificate Serial Number	23764D24759275FA8CC5859D95C85565D11FDB98
Issuer DN	CN=PoliTO Sub CA,O=Politecnico di Torino,C=IT
Valid from	2024-03-06 15:26:35+00:00
Valid to	2026-03-06 15:26:34+00:00
Subject DN	**CN=PoliTO OCSP Responder**
Subject Alternative Name	
Subject Directory Attributes	None
Public key	RSA (4096 bits): ADB9BC7A04DB3F2ED587565B923C7A5B172886E27DA7AA9...
Basic constraints	**End Entity**
Key usage	Digital Signature
Extended key usage	OCSP Signer
Name constraints	No
Authority Information Access	No
Qualified Certificates Statements	No
Certificate Transparency SCTs	No
Signature Algorithm	SHA512WITHECDSA
Fingerprint SHA-256	8FB540F900B1D4DA3EEF5B316B683433 F4A510DB61731B4A681E87E3948E1EC2
Fingerprint SHA-1	B4D4CB32965D1D79DFA89E12A186BC61ABE235FF
Revoked	No

Figure 3.17: Example of OCSP Signer certificate to bind to an OCSP Responder.

3.4.4 Make OCSP requests

In order to make OCSP requests, an OCSP client is required. An open-source solution is OpenSSL, which allows to perform the request executing the command:

```
openssl -issuer <CAcertificate.pem> -CAfile <trustedCAs-chain.pem> -cert
    <CertificateToCheck.pem> -req_text -url <LinkOfTheOCSPService>
```

The option `-issuer` is used to specify the certificate of the CA that issued the certificate in question, while `-CAfile` provides a list of trusted CA certificates for the purpose of verifying the OCSP response. For the previous configuration the two options lead to the same file since the trusted CA is only one and corresponds to the issuer CA, and the URL would be `http://localhost:81/ejbca/publicweb/status/ocsp` if the OCSP request comes from the host machine. Fig. 3.18 and Fig. 3.19 provide two examples of OCSP response.

If the response produces no errors or does not fail, it is possible to disable OCSP protocol CA side. This will block the requests queried directly to the CA from `http://localhost:80/ejbca/publicweb/status/ocsp`, encouraging the use of the dedicated VA. To do so, it is necessary to go to "System Configuration" page, and then to "Protocol Configuration", where it is possible to activate or deactivate access to the supported protocols, among which OCSP.

3.4.5 Transfering CRLs exploiting Secure Copy Protocol

A possible enhancement to the previous configuration can be upgrading the communication channel between the CA and the VA, for instance using a server that supports Secure Copy Protocol (SCP), as shown in Figure 3.20 [26].

Before defining SCP, it is important to understand Secure Shell protocol (SSH), which is useful to control remote network machines and UNIX devices, allowing a client to establish an encrypted remote session via CLI with a server over an insecure channel. To initiate a connection there is a series of steps to ensure that the transmitted data is encrypted:

```
OCSP Request Data:
    Version: 1 (0x0)
    Requestor List:
        Certificate ID:
          Hash Algorithm: sha1
          Issuer Name Hash: F01B192F14B9EBA651009948E18354A6D3569E5B
          Issuer Key Hash: 85BF3EF18D60480E0EE7D7453C8E4BAB4E4F944F
          Serial Number: 7BD72345DBE1153339E93E277273698F9771F65B
    Request Extensions:
        OCSP Nonce:
           041067D1CAD780A6BC0176E703F87362231B
Response verify OK
./certificates/EndEntities/TLS_client_first/s292486.pem: good
        This Update: Mar 8 11:11:28 2024 GMT
```

Figure 3.18: Example of OCSP Request with "good" outcome.

```
OCSP Request Data:
    Version: 1 (0x0)
    Requestor List:
        Certificate ID:
          Hash Algorithm: sha1
          Issuer Name Hash: F01B192F14B9EBA651009948E18354A6D3569E5B
          Issuer Key Hash: 85BF3EF18D60480E0EE7D7453C8E4BAB4E4F944F
          Serial Number: 201B1E2BFDBBE628B565E72A8B121FF6BC3FF00C
    Request Extensions:
        OCSP Nonce:
           0410F4664BDA3995F3CB260209C9312835B1
Response verify OK
./certificates/EndEntities/TLS_client_first/revoked_cert.pem: revoked
        This Update: Mar 8 11:37:48 2024 GMT
        Reason: (UNKNOWN)
        Revocation Time: Mar 8 11:16:13 2024 GMT
```

Figure 3.19: Example of OCSP Request with "revoked" outcome.

1. Version exchange. When the connection begins both client and server exchange their SSH version to make sure they are compatible.

2. Algorithm negotiation. The parties agree on methods for key exchange, encryption and hashing.

3. Key Exchange. In this phase, the two hosts agree on a shared secret using algorithms as Diffie-Hellman or ECDH.

4. Host authentication. At this point the server sends a digital signature created with its private key to the client and this one verifies it, having the server's public key stored inside the `known_hosts` file. Then the client can authenticates itself by means of a password or using public key cryptography, i.e. the server must have stored the client's public key and sends an asymmetric challenge to the client.

5. Session keys creation. Using the shared secret from the key exchange phase, both the client and server generate session keys for encrypting, decrypting and integrity the data transmitted during the session.

6. Encrypted communication. The communication channel is encrypted, so the client can securely control and send data to the host.

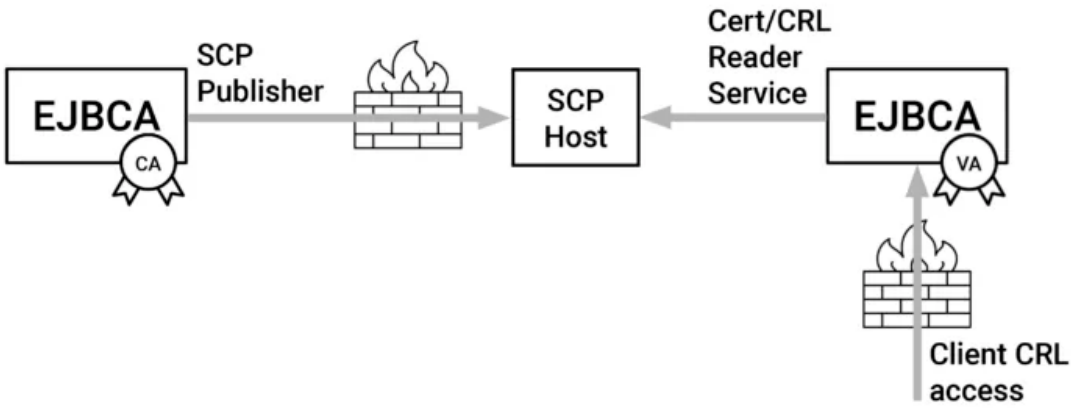

Figure 3.20: Example of architecture where the CA sends CRLs to an external VA via a SCP host (source: EJBCA).

SCP makes use of the SSH protocol to transfer files from a client to a given directory of a remote server. This results in an interesting feature to add to an EJBCA architecture with a distributed VA because this way the CRLs are sent over an encrypted channel, whereas in the previous configuration the download link exploits a HTTP (insecure) connection. To achieve this, it is necessary to configure an SCP-favourable environment, that includes setting up a SCP host, generating the keypair for it and the EJBCA CA instance, creating a network to connect them and eventually establishing trust between the two entities (i.e. public keys exchange). After that, in the Admin Web Page of the CA it is possible to create and arrange a SCP Publisher [27] that will send the CRLs and the certificate to given directories of the server, exploiting the SCP connection. The VA is located in the same network of the SCP Host (or installed into it) and shares the CRLs and certificates directories, and, using the "Certificates and CRL Reader Service" [28], it can populate its database.

3.5 Final remarks and enhancements

At the end of this chapter the obtained result is a PKI featuring 3 main CAs:

- *ManagementCA*: the default CA automatically created at the startup, in charge of issuing certificates for the EJBCA server and the administrator, and managing roles and access rules.
- *PoliTO Root CA*: a custom self signed CA, created for testing. Possibly it can take the place of "ManagementCA".
- *PoliTO Sub CA*: a custom Sub CA, signed by "PoliTO Root CA". It is the issuing Authorithy, configured to issue certificates for the company's End Entities of various kind, depending on company's needs.

Then an external Validation Authority has been configured to distribute the network traffic between the entities who need to issue and manage their certificates, that will address the CA/RA module, and those who want to make OCSP requests, that will turn to the VA. The entire infrastructure has been built to exploit the main features of EJBCA CE and is organised in such a way that each user involved in the organisation has a defined role with its own privileges and restrictions, guaranteeing the least privilege principle. Such configuration that the CAs remain well protected, although RA operations cannot be separated.

In the next chapter it will be explained in detail how the adoption of PKCS#11 devices can enhance the security of a PKI and a virtual HSM will be integrated to the previously designed architecture

Chapter 4

HSM integration

Through this Chapter, PKCS#11 interfaces are going to be analysed in detail and it will be understood why their employment is so important in a PKI ecosystem. Eventually, the security of the previously configured EJBCA-based PKI will be upgraded using an open-source virtual-HSM, explaining the design choices and the procedures to achieve the objective.

4.1 PKCS#11

Among the main Public Key Cryptography Standards (PKCS) introduced in 2.4.3 there is PKCS#11 which sets forth specifications for a platform-independent API for cryptographic devices, such as smart cards, USB tokens and Hardware Security Modules (HSMs), and allows for a high level of flexibility and has become a widely adopted standard for cryptographic device interaction [29].

The API, known as Cryptoki (Cryptographic Token Interface), is implemented in ANSI-C and defines the most commonly used cryptographic object types, such as asymmetric keys or X.509 certificates, and all the functions needed to use, generate, modify and delete those objects. The corresponding modules are developed in C as Dynamically Linked Libraries (`.dll`) or Shared Objects (`.so`), depending on the operating system [30]. It is important to note that PKCS#11 provides an interface to the API, not its features, leaving libraries to detail supported algorithms for specific devices.

In this model, a driver establishes the communication between the cryptographic device, either hardware or software, and the application and realises the commanded operations. After the session is initialised, the application can access token objects and function in Read-Only or Read/Write mode, and there are three main categories of operations that can be performed:

- Administrative, that refers to user's authentication.
- Object management, which include creating, modifying or destroying objects.
- Cryptographic computation, i.e. encryption, decryption, computing digest.

Furthermore, the API supports the separation of access and permissions defining two categories of users:

- *Normal Users*: they are accessible by production applications, so they are allowed to use the full API but they are restricted from administrative actions, such as managing other users, changing global HSM settings, or performing key management tasks that could compromise security. These limitations prevent this kind of user, which is the most exposed one the PKCS#11 model, to perform API-level attacks, i.e. an attacker might exploit a vulnerability in the application to gain access to the PKCS#11 device. From an efficiency perspective, a single session as a Normal User can perform the following set of operations at the same time:

  - message digesting and encryption;
  - decryption and message digesting;
  - decryption and signature verify.

- *Security Officers*: SOs are special users since they do not have access to the full API, but they only perform administrative tasks, such as setting the PKCS#11 module and manage keys or users. SO accounts should be accessed by special management applications or directly by humans and their credentials should never be used by production applications.

Many PKIs, such as EJBCA, adopt PKCS#11 API to access the CA signing key or to enroll certificates.

4.1.1 Cryptoki functions

In order to manage the crypto tokens and the keys inside them, the API has a set of functions that allow to perform the cryptographic operations. Such functions are categorised into different groups [30]:

- *General-purpose functions*: this category of functions governs the overall operational context of the Cryptoki library, facilitating initialisation, cleanup, and information retrieval about the Cryptoki implementation. These functions ensure that the library is correctly prepared for use and can adequately respond with pertinent information regarding available functionalities and interfaces.
  - `C_Initialize`: this function is called to initialise the Cryptoki library. It must be invoked before any other Cryptoki function to prepare the library for operation. This includes setting up any necessary internal structures, checking for the presence of necessary hardware and software resources, and loading any required configuration settings.
  - `C_Finalize`: this function is used to release all resources associated with the Cryptoki library. It should be called after all cryptographic operations are completed, allowing for a clean termination of library operations, including the proper disposal of sensitive information and the de-allocation of memory resources.
  - `C_GetInfo`: provides general information about the Cryptoki library, including the version of the library and details about the manufacturer.
  - `C_GetFunctionList`: returns a list of all the Cryptoki functions that the library supports. This is particularly useful for dynamically linked libraries, allowing applications to discover available cryptographic operations at runtime and adjust accordingly to the functionality supported by the library.
  - `C_GetInterfaceList`: retrieves a list of all the interfaces supported by the library. Each interface may provide different sets of functionalities, tailored for specific cryptographic needs or hardware capabilities.
  - `C_GetInterface`: this function is used to obtain detailed information about a specific interface provided by the Cryptoki library.
- *Slot and token management functions*: this group includes functions responsible for managing and retrieving information about slots and tokens within cryptographic devices. They facilitate interactions with the physical and logical elements of the hardware, allowing applications to perform initial configuration, status checks, and management of user authentication credentials.
  - `C_GetSlotList`: retrieves a list of slot identifiers for all slots connected to the system. This function can be used to enumerate slots which might contain a cryptographic token or be empty, allowing applications to identify available cryptographic resources.
  - `C_GetSlotInfo`: provides information about a specific slot, such as its description, manufacturer, and whether a token is present.

  - `C_GetTokenInfo`: obtains comprehensive information about the token present in a specified slot, as the label, manufacturer ID, model, serial number, and storage capacities.
  - `C_WaitForSlotEvent`: this function waits for a slot event, such as the insertion or removal of a token, to occur on any slot on the system.
  - `C_GetMechanismList`: returns a list of the cryptographic mechanisms – i.e. algorithms and operations – supported by a token.
  - `C_GetMechanismInfo`: provides information about a specific cryptographic mechanism available on a token.
  - `C_InitToken`: initialises a token, setting a label for the token and optionally changing the security officer (SO) PIN.
  - `C_InitPIN`: initialises the normal user's PIN.
  - `C_SetPIN`: modifies the PIN of the user who is logged into the token.

- *Session management functions*: they govern the lifecycle and control of sessions between a client application and a cryptographic token.
  - `C_OpenSession`: initiates a new session between an application and a token in a particular slot.
  - `C_CloseSession`: ends a single session between an application and a token, ensuring proper security hygiene.
  - `C_CloseAllSessions`: terminates all sessions that an application has with a token.
  - `C_GetSessionInfo`: retrieves information about a particular session, such as its state, error codes, and the device it is connected to.
  - `C_SessionCancel`: allows an application to cancel a long-running operation that was previously initiated in the context of the specified session.
  - `C_GetOperationState`: obtains cryptographic operations state in a session.
  - `C_SetOperationState`: sets the cryptographic operations state of a session.
  - `C_Login`: logs a user into a token session, enabling access to objects and operations. Depending on the user type (normal user or security officer), different levels of access and control are granted.
  - `C_LoginUser`: logs into a token with explicit user name
  - `C_Logout`: logs the user out of a token session.

- *Object management functions*: this group handles cryptographic objects within a session, allowing for the creation, copying, modification, and deletion of objects such as keys, certificates, and other data.
  - `C_CreateObject`: creates a new object like a key or a certificate within a token. This function specifies the object's attributes at the time of creation.
  - `C_CopyObject`: generates a duplicate of a specified object.
  - `C_DestroyObject`: permanently deletes an object from a token.
  - `C_GetObjectSize`: retrieves the size in bytes of a specified object.
  - `C_GetAttributeValue`: retrieves one or more attribute values of a specified object.
  - `C_SetAttributeValue`: modifies one or more attributes of a specified object.
  - `C_FindObjectsInit`: initializes a search operation for objects that match a given template.
  - `C_FindObjects`: continues a search for objects.
  - `C_FindObjectsFinal`: completes a search operation initiated by `C_FindObjectsInit`, freeing any resources allocated during the search process.

- *Encryption functions*: this set of function manage the setup, execution, and completion of encryption operations.

  - `C_EncryptInit`: initialises an encryption operation using a specified encryption mechanism and key.
  - `C_Encrypt`: encrypts single-part data.
  - `C_EncryptUpdate`: starts or continues a multi-part encryption operation, processing another data segment.
  - `C_EncryptFinal`: concludes a multi-part encryption operation, cleaning up the encryption context.

- *Decryption functions*: this group performs the cryptographic operations for data decryption. As the encryption functions, there is an initialisation part `C_DecryptInit` where the key and the mechanism is passed; `C_Decrypt` is used for single-part data, while `C_DecryptUpdate` and `C_DecryptFinal` are for multi-part data blocks.

- *Message digesting functions*: compute the digest of a message with the functions `C_DigestInit`, `C_Digest`, `C_DigestUpdate`, `C_DigestKey`, and `C_DigestFinal`.

- *Signing and MACing functions*: include `C_SignInit`, `C_Sign`, `C_SignUpdate`, `C_SignFinal`, `C_SignRecoverInit`, and `C_SignRecover`.

- *Functions for verifying signatures and MACs*: `C_VerifyInit`, `C_Verify`, `C_VerifyUpdate`, `C_VerifyFinal`, `C_VerifyRecoverInit`, and `C_VerifyRecover`.

- *Dual-purpose cryptographic functions*: designed to perform two cryptographic operations in one step, combining processes like encryption and digesting, or decryption and verification. `C_DigestEncryptUpdate`, `C_DecryptDigestUpdate`, `C_SignEncryptUpdate`, `C_DecryptVerifyUpdate`.

- *Key management functions*

  - `C_GenerateKey`: generates a secret key for symmetric encryption algorithms within the cryptographic token.
  - `C_GenerateKeyPair`: generates a pair of asymmetric keys.
  - `C_WrapKey`: encrypts a cryptographic key using a specified wrapping key, usually another key residing within the cryptographic token.
  - `C_UnwrapKey`: decrypts a wrapped key using a specified unwrapping key and imports it into the cryptographic token.
  - `C_DeriveKey`: generates a key from a base key, using a specific key derivation function.

- *Random number generation functions*

  - `C_SeedRandom`: used to seed the random number generator (RNG) of a cryptographic token. It allows an application to provide an external source of randomness, enhancing the entropy of the RNG output, especially when the default seeding mechanism of the token might not be sufficient under certain conditions.
  - `C_GenerateRandom`: generates a sequence of random or pseudorandom bytes based on the seed.

- There are other groups of functions that are designed for message-based operations, i.e. data streams and messages. Such groups are *message-based encryption and decryption functions*, *message-based signature functions*, *message-based functions for verifying signatures and MACs*.

In addition the API supports custom callback functions to alert an application about specific events, and to manage mutex objects, ensuring secure access to the library in multi-threaded environments.

4.1.2 HSM Overview

A PKCS#11-compliant device is the Hardware Security Module (HSM), a physical, tamper resistant machine that serves as a dedicated cryptographic processor. It is engineered to securely generate, store, and manage cryptographic keys that are employed in the encryption and decryption of data, as well as in the creation of digital signatures and certificates [31].

This design effectively isolates these critical cryptographic processes from other system operations, thereby mitigating the risk of exposure to external threats and enhancing the overall security of the system. The key generation is performed using true random number generators.

The security properties guaranteed by HSMs are:

- Confidentiality: the stored keys are encrypted by the module.
- Integrity: since the module is tamper resistant, any attempt to physically access or tamper the HSM is detectable. In many cases, such attempts will trigger self-destruct mechanisms that erase sensitive data.
- Authentication: access to the HSM is controlled through authentication mechanisms like password protection, biometric scanning or smart cards.
- Authorization: many modules can also provide access permissions based on roles, each one with specific privileges.
- Accounting: every action taken within the device is recorded, this is useful for monitoring and reporting purposes.
- Non-repudiation: since the private keys used for signing are securely stored, the HSM ensures that signatures are genuine and traceable to a specific user or entity.

Some of the devices that embrace the definition of "HSM" are smart cards, network encryption devices, CA signing devices, and retail Point of Sales (POS) terminals. Such modules can connect to host computers via bus, like Ethernet or fiber cables, or through smart card readers, keypads etc.

A HSM can be emulated inside a computer using a Virtual HSM software, i.e. an application that provides cryptographic functions and secure key management while leveraging the flexibility and scalability of virtualization. In this Master Degree Thesis Project the chosen framework is SoftHSMv2.

4.1.3 SoftHSM

SoftHSM is an open-source virtual HSM developed by OpenDNSSEC [32], a project whose objective is driving adoption of Domain Name System Security Extensions (DNSSEC) to further enhance Internet security. Initially, this framework was developed for users who did not want to buy a HSM, that could result in an expensive investment, but want to meet the general requirements for DNSSEC, but then it has been found useful for other applications that need access to a PKCS#11 interface.

The tool depends on Botan and OpenSSL cryptographic libraries and its architecture is shown in Figure 4.1 [33].

- *PKCS#11 interface*: the implementation of the Cryptoki interface, version 2.30. All calls from outside the library enter here and are passed on to the other components of the framework.
- *Initialisation and configuration*: is used all through the application to interact with configuration files and is used during the initialisation.
- *Auditing*: provides logging services to other components.

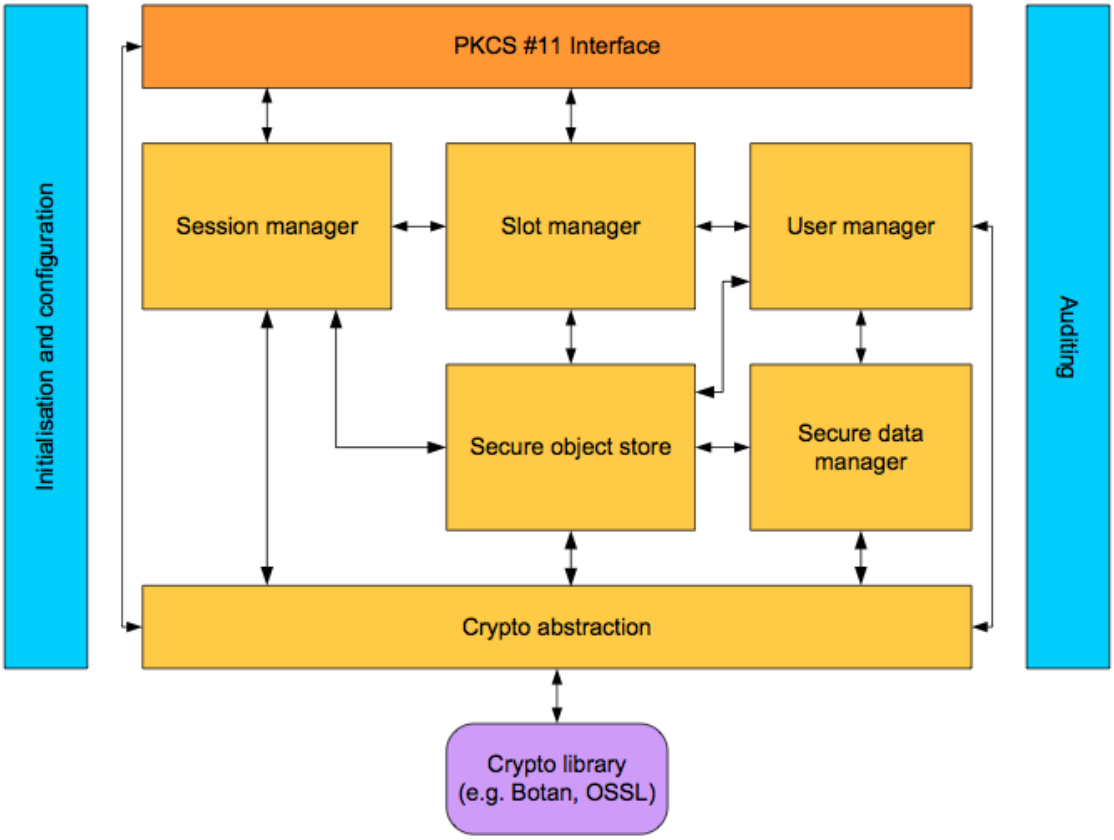

Figure 4.1: SoftHSMv2 Design (source: OpenDNSSEC).

- *Session manager*: manages session objects and keeps track of all the PKCS#11 session and the associated state.
- *Slot manager*: this component manages all PKCS#11 slots and their associated tokens.
- *User manager*: controls the user credentials, i.e. the PIN and the SO PIN.
- *Secure object store*: it is part of the backend storage of SoftHSM2. The PKCS#11 objects are organized in a top-level directory, where each token, identified using a UUID, has its own folder that contains token files (for attributes, label, PINs, etc.).
- *Secure data manager*: derives a key from the user PIN, which is used to encipher/decipher the token key.
- *Crypto abstraction*: it is a layer between an actual cryptographic library (like OpenSSL or Botan) and the framework core.

The choice of the Virtual HSM in this project has led to this software because of its shallow learning curve and its good compatibility with the EJBCA environment [34].

4.2 Integrate SoftHSMv2 with EJBCA CE

To combine this framework with the environment created in Chapter 3, it is necessary to make some configurations and modifications to the default EJBCA CE image. Through this section, it will be shown how to properly set up a HSM which is ideally connected via a peripheral to the application, with whom shares the `token` directory.

4.2.1 Architecture outline

To better understand the structure of the designed PKI, it is necessary to view and understand how the containers are organised (Code 4.1).

```
networks:
  access-bridge:
    driver: bridge
  application-bridge-ca:
    driver: bridge
  application-bridge-va:
    driver: bridge
services:
  ejbca-database-ca:

    # ...

  ejbca-database-va:

    # ...

  ejbca-ca-node:
    hostname: ejbca-ca-node
    container_name: ejbca-ca
    build:
      context: .
      dockerfile: Dockerfile.ejbca
    depends_on:
      - ejbca-database-ca
      - virtual-hsm-ca
    networks:
      # ...
    environment:
      # ...
      - SOFTHSM2_CONF=/opt/softhsm/config/softhsm2.conf
    volumes:
      - ./ca/truststore:/mnt/external/secrets/tls/ts
      - ./ca/hsm-conf:/opt/softhsm/config:ro
      - ./ca/tokens-ca:/opt/softhsm/tokens:rw
    ports:
      # ...
  ejbca-va-node:
    hostname: ejbca-va-node
    container_name: ejbca-va
    build:
      context: .
      dockerfile: Dockerfile.ejbca
    depends_on:
      - ejbca-database-va
      - virtual-hsm-va
    networks:
      # ...
    environment:
      # ...
      - SOFTHSM2_CONF=/opt/softhsm/config/softhsm2.conf
    volumes:
      - ./va/secrets-va/truststore:/mnt/external/secrets/tls/ts
      - ./va/hsm-conf:/opt/softhsm/config:ro
      - ./va/tokens-va:/opt/softhsm/tokens:rw
    ports:
      # ...
  virtual-hsm-ca:
```

```
    container_name: virtual-hsm-ca
    build:
      context: .
      dockerfile: Dockerfile.softhsm
    volumes:
      - ./ca/tokens-ca:/var/lib/softhsm/tokens/:rw
  virtual-hsm-va:
    container_name: virtual-hsm-va
    build:
      context: .
      dockerfile: Dockerfile.softhsm
    volumes:
      - ./va/tokens-va:/var/lib/softhsm/tokens/:rw
```

Code 4.1: Docker Compose configuration to integrate the virtual HSMs to the PKI.

The omitted parts are kept unchanged with respect to the previous version (Code 3.2). The things to notice in this design are:

- The CA and the VA have their own independent module, so that CAs' keys are preserved in case of attack or malfunctioning in the VA.
- In the folder `./ca/hsm-conf/` (and `./va/hsm-conf/`) is contained the `softhsm2.conf` file, which indicates the position of the token directory seen from the EJBCA point of view, i.e. `/opt/softhsm/tokens/`. The environment variable `SOFTHSM2_CONF` specifies the location of the configuration file to the application.
- The Virtual HSM containers are not connected to a network but they need to start before EJBCA, to load the tokens for the application. This is why EJBCA services need to depend on their HSMs, as well.
- The tokens directory in the HSM and in EJBCA point to the same host directory. This simulates a shared tokens folder.

Now it is time to understand what is the content of `Dockerfile.ejbca` (Code 4.2) and `Dockerfile.softhsm` (Code 4.3):

```
FROM centos:7 AS builder

# Install dependencies required to build OpenSSL 3
RUN yum -y update && \
    yum -y groupinstall "Development Tools" && \
    yum -y install perl-IPC-Cmd wget

# Download and extract OpenSSL 3
WORKDIR /tmp
RUN wget https://www.openssl.org/source/openssl-3.2.1.tar.gz && \
    tar xvf openssl-3.2.1.tar.gz

# Build and install OpenSSL 3
WORKDIR /tmp/openssl-3.2.1
RUN ./config --prefix=/usr/local/openssl --openssldir=/usr/local/openssl && \
    make -j $(nproc) && \
    make install

# Update the shared libraries cache and system-wide OpenSSL configurations
RUN ldconfig && \
    echo 'export PATH=/usr/local/openssl/bin:$PATH' >>
        /etc/profile.d/openssl.sh && \
```

```
    echo 'export
        LD_LIBRARY_PATH=/usr/local/openssl/lib:/usr/local/openssl/lib64
        :$LD_LIBRARY_PATH' >> /etc/profile.d/openssl.sh

# Set environment variables to use the new OpenSSL version
ENV LD_LIBRARY_PATH="/usr/local/openssl/lib64:${LD_LIBRARY_PATH}" \
    LDFLAGS="-L/usr/local/openssl/lib64 $LDFLAGS"

# Continue with your previous steps
WORKDIR /build
COPY ./hsm-conf/softhsm2.conf /etc/softhsm2.conf

# Install additional dependencies required for your application
RUN yum -y install automake autoconf libtool pkgconfig && \
    curl --silent -D - -L
        "https://github.com/opendnssec/SoftHSMv2/archive/refs/tags/2.6.1.tar.gz"
        -o SoftHSMv2-current.tar.gz && \
    mkdir -p /build/softhsmv2 && \
    tar -xf SoftHSMv2-current.tar.gz -C "/build/softhsmv2"
        --strip-components=1 && \
    cd softhsmv2/ && \
    ./autogen.sh && \
    ./configure --disable-non-paged-memory --prefix=/usr
        --with-openssl=/usr/local/openssl && \
    make -s && \
    make -s install && \
    # Create the directory for the tokens
    mkdir -p /var/lib/softhsm/tokens && \
    chgrp -R 0 /var/lib/softhsm/ /etc/softhsm2.conf && \
    chmod -R g=u /var/lib/softhsm/ /etc/softhsm2.conf && \
    chown -R 10001:0 /var/lib/softhsm/tokens

FROM keyfactor/ejbca-ce:latest

USER 0:0

COPY --from=builder --chown=10001:0 /usr/local/openssl/lib64/libcrypto.so.3
    /usr/lib64/libcrypto.so.3
COPY --from=builder --chown=10001:0 /usr/lib/softhsm/libsofthsm2.so
    /usr/lib/softhsm/libsofthsm2.so

USER 10001:0
```

Code 4.2: `Dockerfile.ejbca`

```
FROM centos:7 AS builder

# Same builder as in Dockerfile.ejbca

FROM centos:centos7

COPY --from=builder --chown=10001:0 /usr/local/openssl/ /usr/local/openssl/
COPY --from=builder --chown=10001:0 /var/lib/softhsm/ /var/lib/softhsm/
COPY --from=builder --chown=10001:0 /usr/lib/softhsm/libsofthsm2.so
    /usr/lib/softhsm/libsofthsm2.so
```

```
COPY --from=builder --chown=10001:0 /usr/bin/softhsm2-keyconv
    /usr/bin/softhsm2-keyconv
COPY --from=builder --chown=10001:0 /usr/bin/softhsm2-util
    /usr/bin/softhsm2-util
COPY --from=builder --chown=10001:0 /usr/bin/softhsm2-dump-file
    /usr/bin/softhsm2-dump-file
COPY --from=builder --chown=10001:0 /etc/softhsm2.conf /etc/softhsm2.conf

# Set the environment variable for the configuration file
ENV SOFTHSM2_CONF="/etc/softhsm2.conf" \
    PATH="/usr/local/openssl/bin:$PATH" \
    LD_LIBRARY_PATH="/usr/local/openssl/lib:
        /usr/local/openssl/lib64:${LD_LIBRARY_PATH}" \
    LDFLAGS="-L/usr/local/openssl/lib64 $LDFLAGS"

USER 10001:0

CMD ["tail", "-f", "/dev/null"]
```

Code 4.3: `Dockerfile.softhsm`

The behaviour of the two files is analysed below.

- Both Dockerfiles use the same builder which installs SoftHSM and its dependencies. The configuration file `./conf-hsm/softhsm2.conf` imported at the beginning sets `/var/lib/softhsm/tokens/` as default token directory.
- `Dockerfile.ejbca` only imports the library files `libsofthsm2.so` and `libcrypto.so.10`, i.e. the drivers to read the tokens, in the EJBCA image. The destination directories are not a case, in fact the application recognises the HSMs only if their libraries are located in specific folders, specified in the file `/opt/keyfactor/ejbca/conf/web.properties`.
- `Dockerfile.softhsm`, imports all the files needed to run and manage SoftHSM.
- The default user in both the containers is set to `10001:0`, which is the entity that has the rights to run EJBCA and executes the commands for the application. Configuring the same user into the HSM container would not create problems during the manipulation of the tokens in the shared directory.

4.2.2 Create and manage tokens

After building the containers, it is time to run and test the HSM functions. To create a new token for testing, it is necessary to control the CLI of a HSM container, e.g. `virtual-hsm-ca`, running the `docker exec` instruction from the host machine and, once inside, executing the following command:

```
softhsm2-util --init-token --free --label <token_name> --pin
    <alphanumeric_pin> --so-pin <alphanumeric_so_pin>
```

Where the option `--init-token` starts the initialization process, `--free` select the first free memory slot to save the token, `--label` allows to give a name to the token, then `--pin` and `--so-pin` set the PINs to access either in User or in SO mode. Then, it is possible to see the created tokens with the command:

```
softhsm2-util --show-slots
```

And finally, the instruction to correctly remove a token is:

```
softhsm2-util --delete-token --token <label>
```

Figure 4.2: New Crypto Token page when selecting "PKCS#11" option as "Type".

After the token creation, it is important to restart EJBCA in order to be able to interact with it; in fact, the server only loads the tokens during the startup. To add a PKCS#11 token, it is necessary to go to the "Crypto Tokens" page, in the Admin Web of the application, and then go to "Create new...". Here (Fig. 4.2):

- In "Type", select the option "PKCS#11" and, if the configuration is successful, the "SoftHSM 2" driver will be automatically selected as "PKCS#11 Library".
- As "PKCS#11 : Reference Type", there are various possibilities to recognise the wanted token, the easiest way is selecting "Slot/Token Label".
- In "PKCS#11: Reference" a drop down menu will appear if the reference type is "Slot/Token Label" where it is possible to choose the token from its label.
- The "Authentication Code" corresponds to the User PIN chosen at the moment of the token creation.

After saving the changes, it is possible to generate the keypairs inside the new PKCS#11 Crypto Token from the web page; the behaviour is identical to a keypair generation using a Soft Crypto Token, but the difference is that the key generation, storage and management are up to SoftHSM.

4.2.3 Final steps

At this point, there are two ways to conclude the configuration, a first one would be migrating the previously created keys to the HSM slots. This choice gives a false sense of security because if the keys are generated outside the HSM, there is not guarantee that they are not copied around already. Plus, the key generation performed by a real HSM is safer because it exploits High Entropy Number Generators (not the case of SoftHSM though).

The best practice would be recreating a PKI from scratch using the same schema proposed in the previous Chapter, this time with the key generated inside the HSM. In this case it is possible to apply this solution because the PKI was just a prototype and not in service; but if a PKI is online and the CA produces high amount of certificates, the only way would be migrating from a HSM to another copying the keys securely.

Chapter 5

PKI Maintenance, Performance and Security

This Chapter focuses on the aspects that need to be taken into account when managing a PKI in order to keep it robust over time. Such features allow for the assurance of security, performance, and reliability within a digital environment that increasingly demands high standards of trust and operational efficiency. The following sections provide the guidelines on best practices for maintenance, the evaluation of system performance under varying loads, and the measures to fortify security against evolving threats. Eventually, real-world use cases of EJBCA CE are presented, illustrating the practical application and versatility of the designed PKI in addressing various requirements.

5.1 Maintenance

Maintenance is crucial in a PKI ecosystem; each component of a PKI need constant support after their implementation. Upkeep operations include, but are not limited to, subjecting modules to periodic updates, cleaning the databases, and generally optimizing the whole system to ensure efficient activity. Indeed, regular maintenance not only aids in enhancing the security posture by addressing newly discovered vulnerabilities, but also plays a pivotal role in ensuring the scalability and flexibility of the PKI to adapt to evolving technological landscapes and organizational needs.

This section will encompass strategies for regular system audits in the context of EJBCA CE and the importance of adhering to compliance standards. Finally, it will highlight the challenges in maintaining a PKI, such as managing the lifecycle of certificates, ensuring the continuity of trust, and balancing between system updates and operational uptime.

5.1.1 System Auditing and Monitoring

In order to understand what is happening inside the EJBCA server, the operator has access to the audit logs, which notify in a human way all the events occurred during the execution of the container. This feature is really important to monitor the internal workings of the framework, and to realise where the issue stands during a troubleshooting session.

In the Docker environment, there are four types of log messages:

- `INFO`: documents the normal and expected operations of the system. This kind of message is beneficial for general monitoring and operational reviews, providing a clear picture of the system's activity.
- `DEBUG`: if activated, provides more verbose information about an event, offering granular details about the system's state and the lower level steps that are being executed, which are

essential for diagnosing complex issues. They are particularly useful during developing and to ensure that all the components are functioning as intended.

- `WARN`: a warning message that highlights something unexpected, which is not necessarily an error but could potentially cause one if ignored.
- `ERROR`: generally signals that a specific operation could not be completed as expected due to some form of error or failure. Although these issues are serious, they do not necessarily halt the overall operation of the application or system; rather, they highlight areas that require attention or intervention to prevent further complications or to restore functionality.
- `FATAL`: records severe issues that cause the system to terminate or stop functioning altogether. It indicates critical failures where recovery is not possible without intervention.

The command to view the audit logs is:

```
docker compose logs <service-name> -f
```

Where `service-name` is optional and, if omitted, the logs of all the active services will flow.

Inside the container, from the framework point of view, two types of events can be recognised, CORE and EJBCA; both are associated to the EJBCA application, but CORE contains functions also shared with other products. To interpret a log message, it is important to understand how it is structured [35], analysing all its columns:

- *Service*: The service an event originates from, `EJBCA` or `CORE`.
- *Module*: The module an event was generated from. Example of modules are `CA`, `RA`, `ACCESSCONTROL`, `AUTHENTICATION`, `CRL` etc.
- *Status*: Indicates the outcome of the event `SUCCESS` (operation succeeded), `FAILURE` (operation failed) or `VOID` (Operation completed without a defined result).
- *Event*: The audit log event that occurred. If `EJBCA` service is involved the event would be a `EjbcaEventType`, instead, in case of `CORE` service the event will be a `CESeCore` Event:
- *AdditionalDetails*. Event specific message with additional information.
- *AuthToken*: Identifies the administrator, or internal module, that caused the event.
- *NodeId*: Identifies the EJBCA instance (which server in a cluster) that the event occurred on.
- *CustomId*: Identifier used in log messages, commonly the certificate authority an event was related to.
- *searchDetail1*: Detail used in log messages, commonly the serial number of the certificate an event was related to.
- *searchDetail2*: Detail used in log messages, commonly the username an event was related to.
- *timeStamp*: The time, in milliseconds since epoch, the event occurred.

Another tool useful to monitor the status of the PKI server is the Health Check [36]. In EJBCA CE, it is possible to activate the Health Check for each CA in "CA Activation" page, located in the sidebar menu of the Admin Web page, and then access via the servlet `http://localhost:80/ejbca/publicweb/healthcheck/ejbcahealth`. In the basic configuration, if all the monitored CAs are active and working the response will be `ALLOK`, instead, if at least one CA is inactive or malfunctioning the displayed message will be `EJBCANOTOK`. It is possible to customise the responses and the messages of the servlet from the file `/opt/keyfactor/ejbca/conf/ejbca.properties`. Finally, to further enhance the supervision of the entire infrastructure, it is possible to set up an open-source tool called Monit [37] and implement an observer server. This application provides a UI that shows information about the status of the server, the database, the CPU and memory usage etc. (Fig. 5.1) [38].

System	Status	Load	CPU	Memory	Swap
localhost[alma90]	OK	[0.01] [0.03] [0.04]	0.5%us 0.2%sy 0.0%ni 0.0%wa	60.5% [2.1 GB]	0.0% [0 B]

Process	Status	Uptime	CPU Total	Memory Total	Read	Write
MariaDB	OK	12m	0.1%	2.8% [99.8 MB]	463.9 B/s	272.7 B/s
HTTPD	OK	12m	0.1%	2.1% [75.2 MB]	0 B/s	0 B/s
Wildfly	OK	12m	1.9%	44.7% [1.6 GB]	0 B/s	0 B/s

Program	Status	Output	Last started	Exit value
EJBCA[Healthcheck]	OK	EJBCA is OK!	29 Nov 2022 00:07:26	0

Filesystem	Status	Space usage	Inodes usage	Read	Write
localhost[/boot]	OK	30.9% [312.8 MB]	0.1% [374 objects]	0 B/s	0 B/s
localhost[/root]	OK	34.6% [5.9 GB]	0.9% [76268 objects]	82.6 kB/s	1.5 kB/s

Net	Status	Upload	Download
Interface[Bridge]	OK	868 B/s	264 B/s

Figure 5.1: Monit UI (source: PrimeKey).

5.1.2 Certificate Lifecycle Management

To guarantee the security and the integrity of the PKI ecosystem, it is important to accept that the issued digital certificates do not last forever, but they follow a series of protocols and rules that are encompassed inside the Certificate Lifecycle Management; in fact, once a certificate has been issued it must be managed carefully through its entire validity period and this is not a trivial task. Some of the procedures of the Certificate Lifecycle Management have already been seen in the previous Chapters and include the following stages (Fig. 5.2) [39]:

Figure 5.2: Stages of Certificate Lifecycle Management (source: Keyfactor).

- *Discovery*: during this process, the entire network infrastructure is reviewed to determine where each certificate is installed and to verify if it has been implemented properly. This

analysis corroborates the environment only includes known certificates, configured with the right protocols. This results in an actual inventory that correlates a digital certificate to its devices and eases renewal and revocations operations.

- *Generation*: in this phase it is really important to keep clear the purpose for which the certificate is going to be issued, i.e. the Key Usage and the Extended Key Usage, so choosing the right algorithms and giving the proper information is crucial. As seen in the previous Chapter the applicant must send a CSR that includes its identifying details and eventually sign the request with the private key. The Authority then will possibly process the request and build the certificate according to the received data.

- *Analyse*: in complex architectures with many types of certificates, the certificate management becomes harder; this is why it is important to employ a figure that must handle various certificate formats, each with specific requirements for metadata properties like encryption key length and cryptographic algorithms.

- *Supervision*: the validity of a certificate may vary based on its type; the standard is 398 days, which promotes an annual maintenance with some tolerance. The primary focus in managing certificates involves handling keys and certificates through protocols that facilitate submission, updating, backup, restoration, and revocation, and must be compatible across various client applications and the Certification Authority. Some systems, like EJBCA, allow for automatic renewal of certificates before they expire. For an effective certificate management it is essential to oversee all the certificate chain and use custom fields for storing additional data.

- *Monitor*: this stage is useful to help administrators renew a certificate before expiration. There are dynamic monitoring and reporting systems that quickly provide the administrator information about the number of certificates issued by each CA, and, among those, which need to be renewed o replaced. In addition, such systems are capable of notify in advance, for instance via e-mail, if a given certificate is about to expire.

- *Validation*: as seen in the previous Chapter, authentication is checked at the time of transaction by verifying the certificate's legitimacy with the CA or a server, usually via an online process, like OCSP. If a certificate has been revoked, the receiver will not proceed with the transaction based on it. Each certificate has a defined validity period, marked by a start and end date and time, and depends on factors like the strength of the private key signing the certificate or the cost of the certificate. This duration is typically how long entities can trust the public value, assuming the private key remains secure. A certificate might become invalid before its expiry due to various reasons, such as a name change, changes in the relationship between the subject and CA (like an employee leaving a company), or a compromise (or suspected compromise) of the private key. In such instances, the CA is required to revoke the certificate.

- *Revocation*: this is a key step in Certificate Lifecycle Management, that is cancelling the validity of a certificate before its expiration date due to unforeseen circumstances. The revocation methods have already been seen and include the periodic issuance of CRLs. It is important to properly manage this stage because on one hand CRLs can be distributed securely under any channel, but on the other hand the CRL will not be modified until its expiration date; that's why there exist mechanisms to mitigate this drawback, such as issuing smaller data structures like Freshest CRLs or delta CRLs that contain updated informations on the revoked certificates during the activity period of the main CRL.

An adequate certificate management is crucial because, in case of security incidents, accelerates the process of resolution and reduces devices downtime; this approach not only helps avoid additional management costs and but also does not compromise the trust and the reliability of the company.

5.1.3 Update and Patch Management

Keeping a system up to date is really important to guarantee its security and integrity, but it is not an easy practice, especially when the infrastructure is complex and there are many variables to consider. Server updates are indeed delicate operations that expose the architecture to several risks, like data losses and service interruptions, and for this reason there are guidelines to follow during this circumstance.

In general, inside a company the best practice are:

- Evaluate the updates in a test environment before deploying them to endpoints, to check if the new released version does not contain any error and can be safely integrated in the system.
- Schedule updates on a regular basis, for instance on a chosen day of the week, to avoid bandwidth congestion during the whole working time.
- For an efficient patch management, prioritising the updates is useful, handling the critical ones first.
- Backups before the updates are important because, in case incidents, they allow to revert the infrastructure back to its previous stable state with a rollback mechanism.
- To minimize service interruptions, implementing redundancy in server modules is crucial. As outlined in Section 3.3.4, redundancy ensures that services continue uninterrupted, even during updates or maintenance. In patch management, redundancy allows for patches to be applied to one part of the system at a time, while other redundant parts handle the load, and any potential issues caused by new patches can be isolated and addressed without affecting the overall system availability.

Another crucial element to consider within a corporate environment is the change management, which consists in scheduling and documenting all modifications made to each hardware and software module of the infrastructure. This practice is essential for maintaining a well-organised system, facilitating the identification of potential issues and their causes during troubleshooting. By maintaining comprehensive records of system changes, organizations can streamline the diagnostic process, quickly pinpointing which updates or changes may have contributed to a system malfunction.

In the context of PKIs, managing updates embraces more than just software and hardware enhancements; it also critically involves the continual revision and adaptation of internal policies, which govern not only the technical operations but also the compliance standards of the PKI system within the organization. Furthermore, in environments where cross-certification with other companies is necessary to facilitate trusted inter-company communications, it becomes essential to align these policies with external partners. This alignment ensures that all parties adhere to mutually agreed-upon security standards and operational procedures, which might include adherence to broader regulations like PCI DSS, HIPAA, or GDPR, depending on the sector and geography of operation. Regularly updating these policies helps maintain a secure, interoperable, and compliant infrastructure that supports not only internal needs but also collaborative efforts across different corporate entities.

To update EJBCA CE within Docker containers, it is sufficient to execute two commands from the host's working directory:

```
docker compose pull
```

and

```
docker compose build
```

The first one will pull the latest version of the images mentioned in the file `docker-compose.yml`, while the other one will update the content of the Dockerfiles. Beside this, systems upgrades in

every production context are more than simply running a command, they indeed require attention and general knowledge of the whole infrastructure, because there might be components that are not compatible with the newer versions and, if installed, can cause partial or total system malfunctioning. In these cases, it is wise to wait for an update of those elements before proceeding with the general upgrade, ensuring each module will function harmoniously with the new release.

5.1.4 Database Maintenance

The PKI database is another sensitive component that needs maintenance and attention, since it stores the settings and the frameworks that govern the entire ecosystem, CA data, certificates and CRLs informations, and in some cases, although it is strongly discouraged, also End Entities private keys, if key recovery is enabled. EJBCA inside a Docker Container would be ephemeral if not paired with a database, i.e. data during the session is not stored anywhere and will be deleted when the instance is closed; the database employed in this Project utilises an open-source Relational Database Management System (RDBMS) called MariaDB, which originates from a fork of MySQL and provides a fast, scalable and robust server, fully compatible with the framework [40].

In order to understand how to maintain the EJBCA database it is important to know the most important tables of its structure:

- `CAData`: contains informations about the PKI CAs, such as the `caID` an internal CA identifier, the CA certificates `data` in `xml` format, `expireTime`, `name`, `subjectDN` etc.
- `CRLData`: keeps track of all the issued CRLs, and some of its columns are `fingerprint`, `base64Crl`, `cAFingerprint`, `cRLNumber`, `issuerDN`, a flag that indicates whether the CRL is a delta or not, the scheduled `nextUpdate` etc.
- `CertificateData`: this table stores all the certificates that have been issued in the PKI; the stored information include the `issuerDN`, their status (valid, revoked or expired), the Certificate Profile and the End Entity Profile that have been used, the `username` etc.
- `CertificateProfileData` and `EndEntityProfileData`, which respectively contain Certificate and EndEntity Profiles.
- `CryptoTokenData`: the table that includes the Crypto Tokens. In case the entry is a Soft Crypto Token, `tokenData` contains the token keys, stored in a PKCS#12 file encrypted with a password and encoded in base64. Whereas, if it is a PKCS#11 token, that column would be empty and `tokenProps` column indicates the token slot of the HSM it points to.
- `KeyRecoveryData`, that contains the End Entities private keys, if applicable.
- Other tables are `GlobalConfigurationData`, `OcspResponseData`, `PublisherData`, `RoleData`, `RoleMemberData`, `ServiceData`, `UserData` etc.

EJBCA also supports an integrity protection mechanism for its database, which guarantees that data has not been modified by unauthorised entities. The technique consists in using an additional `rowProtection` column in all protected tables, a string that includes embedded version and `keyId`, enabling table modifications and legacy data verification through versioning, i.e. new versions can incorporate additional fields while still validating older entries. The `keyId`, defined by the user, links to a specific cryptographic token for verification, supporting multiple active keys simultaneously for increased flexibility and security. It is suggested to set database protection before EJBCA is installed and used the first time, otherwise the data inserted before the configuration will remain unprotected. However, database protection also has some limitations, such as bulk updates cannot be performed, and direct updates to database are forbidden.

The tables that need particular attentions are `CRLData`, `CertificateData` because they are the ones that will occupy more disk space over time. Often, especially in the context of a big company, it is necessary to delete those records of certificates and CRLs that have been revoked or expired for a certain period, the standard is usually 180 days. This practice keeps the system lightweight and guarantees that no performance degradation will affect it.

5.1.5 Support and Community Resources

Framework documentation and tutorials are critical resources when implementing and maintaining a service. EJBCA CE, in particular, boasts of a detailed documentation and benefits from an active community of PKI experts who provide various types of support. Within a corporate environment, it is imperative that maintenance staff periodically consult these resources to stay remain informed. Furthermore, it is advisable that they document how the infrastructure was developed, explaining design choices and settings, while referencing to the original documentation. Where existing documentation is insufficient, additional reports should be compiled to fill these gaps. This practice ensures long-term maintenance viability, preserving crucial details that might otherwise be lost over time. Additionally, in the event of personnel changes, new team members can acquaint themselves with the system architecture simply by reviewing these resources, thereby facilitating continuity.

The references of interests are:

- EJBCA Documentation: available at https://doc.primekey.com/ejbca, provides comprehensive guidelines and operational procedures.
- Keyfactor Community YouTube Channel: contains playlist with tutorials for beginners that approach the framework for the first time. https://www.youtube.com/@KeyfactorCommunity/
- EJBCA CE DockerHub: accessible at https://hub.docker.com/r/keyfactor/ejbca-ce, offers insights into the integration of the framework within a containerized environment.
- EJBCA CE GitHub: found at https://github.com/Keyfactor/ejbca-ce, where users can engage with developers in the discussions section for troubleshooting and support.
- SoftHSM Wiki: at https://wiki.opendnssec.org/softhsm/, serves as a repository of knowledge and technical support for SoftHSM implementations.

5.2 Performance

EJBCA is engineered to process a large number of certificates under significant transaction loads [41]. In general, a PKI does not require extreme performance, especially in smaller-scale deployments such as the one designed in this project. A default EJBCA installation is capable of issuing certificates at a rate of tens to hundreds per second, which is sufficient for most operational needs.

This section will evaluate the performance of the PKI designed for this project, focusing on certificates issuance rate by the CA, and the responsiveness of the VA in handling OCSP requests. Additionally, the behaviour of SoftHSM will be analysed both in isolation and when integrated with EJBCA's "filter" mechanism. The aim is to assess not only the stand-alone performance of these components but also their efficiency when operating within the orchestrated PKI environment.

In order to comprehend the outcomes that will be obtained in the next tests, it is essential to consider the specifications of the host machine:

- The physical machine has a dual-core "Intel(R) Core(TM) i3-8130U" CPU, an "Intel(R) UHD Graphics 620" GPU and 20 GB of RAM.
- The virtual host machine is configured to run CentOS 7 Linux and the following resources have been allocated to it:
  - Two CPU cores
  - 8 GB of RAM
  - 64 MB of video memory

5.2.1 Stress testing using EJBCA Client Toolbox

The installation of the EJBCA framework also includes a set of tools that provide commands for various functions, including Web Service RA operations, managing keys on PKCS#11 HSM, performing OCSP requests, and executing health checks. This program is a shell executable file called `ejbcaClientToolBox.sh`, located in the `/opt/keyfactor/ejbca/dist/clientToolBox/` directory of the EJBCA containers. It can also be installed as a stand-alone package to run on any machine independently of EJBCA [42].

In general, this toolbox can be utilised to implement scripts that will enhance automation and flexibility in the PKI, but for this specific scenario it will be exploited to stress the infrastructure. To do so, the main arguments to call are `EjbcaWsRaCli` and `OCSP`, the first one serves to issue certificate using the RA web-service, the other performs OCSP requests, and they both support the subcommand `stress`.

5.2.2 RA Web Service stress test

To issue a high number of certificates with `EjbcaWsRaCli`, it is necessary to control the CA container's shell as a root, go to the aforementioned folder and modify the file `ejbcawsracli.properties`. Here the URL of the Web Service must be specified, along with the path to the PKCS#12 file used to authenticate the client to the Web Service (for instance the SuperAdmin certificate), and the password to unlock the keystore, otherwise it will be requested after entering the command. The script accepts the following syntax:

```
stress <caname> <nr of threads> <max wait time in ms to fetch cert after
    adding user> [<end entity profile name>] [<certificate profile name>]
    [<type of test>] [<nr of tests>]
```

where the issuing CA name must be passed as `<caname>`, `<nr of threads>` indicates how many concurrent threads will perform the request, and `<max wait time in ms to fetch cert after adding user>` represents the time a thread should be wait, at most, before proceeding with a new request. The non-mandatory options indicate respectively the End Entity Profile to use, the Certificate Profile, the type of test and the number of test before the script ends. There are five types of tests: `BASIC`, which is the default one, `BASICSINGLETRANS`, `BASICSINGLETRANS_SAMEUSER`, `REVOKE REVOKE_BACKDATED`, and `REVOKEALOT`. The most pertinent ones are:

- `BASIC`: performs a sequence of operations that are typical in certificate management. The process begins with the creation of a user through the `editUser` call, after which a certificate request is initiated via the `pkcs10Request` call. The test simulates typical user interaction with the PKI system, with the objective of assessing the system's capability to handle multiple, sequential operations efficiently. This is is the one that will be used for the assessment.
- `BASICSINGLETRANS`: simplifies the certificate request process by combining the user modification and certificate request into a single API call, `certRequest`. This results in a reduction in the number of network round trips, thereby enhancing the overall system response times and throughput. This one is particularly useful for testing the system's performance under optimal conditions.

In order to test the designed PKI, the thread number has been set to two, as there are only two CPUs, the issuing CA is "PoliTOSubCA", and the total number of issued certificates per test is 100.

```
./ejbcaClientToolBox.sh EjbcaWsRaCli stress "PoliTOSubCA" 2 <x> EMPTY ENDUSER
    BASIC 100
```

the tests were been repeated for different values of maximum wait time, denoted by the variable `<x>`, to evaluate the system's performance under different operational load and concurrency conditions.

Max wait time [ms]	*Tests/Sec. avg*	*% time waiting*	*% time registering users*	*% time signing certificates*	*% time client work*	*Min – Max time reg. users* [ms – ms]	*Min – Max time signing* [ms – ms]
2	18.81	0.91%	33.77%	44.14%	21.18%	13 – 454	16 – 214
3	19.74	1.62%	38.91%	41.23%	18.24%	9 – 546	18 – 194
5	18.11	4.01%	36.40%	43.11%	16.48%	11 – 281	16 – 201
10	13.12	5.97%	37.78%	37.86%	18.39%	14 – 1066	22 – 279
50	15.58	37.75%	25.04%	26.03%	11.18%	9 – 418	14 – 96
100	11.16	55.75%	14.56%	15.17%	14.52%	8 – 353	13 – 130
250	6.37	79.68%	7.81%	9.03%	3.48%	10 – 201	13 – 95
500	3.38	86.67%	5.55%	5.61%	2.17%	10 – 259	13 – 129
750	2.46	90.07%	4.12%	3.95%	1.86%	10 – 431	14 – 73
1000	1.68	81.15%	3.08%	3.32%	12.45%	10 – 244	16 – 432

Table 5.1: Results of the RA Web-Service stress test.

The results, as detailed in Table 5.1, demonstrate a clear trend: as the maximum wait time increases, the average number of tests completed per second consistently decreases. This indicates that higher wait times significantly reduce system throughput, affecting the overall efficiency. It is noteworthy that the proportion of time spent waiting between jobs increases substantially with longer wait times, which dominates the system's activity and leads to decreased operational efficiency. Conversely, at lower wait times, the system exhibits higher throughput and better resource utilisation, managing tasks more efficiently with less idle time. These findings underscore the importance of optimising wait times to balance system responsiveness with workload management, ensuring that the system can handle varying degrees of load without compromising on performance.

5.2.3 OCSP stress test

The tool `EjbcaWsRaCli` generates the file `result.ser`, a Java data structure containing information about the dummy certificates that have been produced during the stress test. This file can be passed as an argument to the `OCSP stress` tool, which will execute numerous OCSP requests will be executed on the previously created certificates. In order to evaluate the performance of the VA, it is necessary to import the aforementioned file, along with the CA Certificate, then control the the CLI of the VA container and reach the script folder. This time, the command follows this structure:

```
stress <OCSP URL> <Certificate serial number file> <ca cert file> <number of
    threads> <max wait time between requests>
```

The argument names are self-explanatory. To run the stress test the user must provide the following information: the link to the OCSP service, the certificate serial number file (either `result.ser` or a text file containing a certificate serial number in hexadecimal form for each row), the CA certificate file, the number of threads and the maximum wait time between requests.

In this specific assessment, it was found that OCSP operations require less resources than certificate generations, allowing for the calling of more than two concurrent threads without a memory fault. Consequently, the stress test was designed to increment the number of threads progressively, simulating increasing loads to observe the VA's capacity and performance thresholds. The aim was to receive the highest possible number of successful responses within 60 s and 2 ms of maximum wait time. The following command was used to execute the OCSP stress test:

```
./ejbcaClientToolBox.sh OCSP stress
    http://ejbca-va-node:8080/ejbca/publicweb/status/ocsp result.ser
    PoliTOSubCA.pem <x> 2
```

Thread #	*Tests/Sec. avg*	*% time waiting*	*% time OCSP lookup*	*% time client work*	*Total # of tests*
2	55.53	1.38%	85.29%	13.33%	3334
3	57.18	0.95%	81.24%	17.81%	3450
4	58.06	0.73%	75.70%	23.58%	3485
5	**59.75**	**0.61%**	**71.25%**	**28.14%**	**3588**
6	57.98	0.48%	66.68%	32.84%	3482
7	55.82	0.41%	62.98%	36.61%	3350
10	55.47	0.27%	53.58%	46.14%	3329
25	53.71	0.11%	48.45%	51.44%	3225
50	52.87	0.05%	55.41%	44.54%	3189
100	52.24	0.03%	85.05%	14.92%	3195
200	37.28	0.01%	92.70%	7.29%	2305

Table 5.2: Results of the OCSP stress test.

where the variable `<x>` represents the different thread numbers for each call.

Table 5.2 indicates that the VA functions optimally at a moderate level of concurrency, with five threads demonstrating the optimal balance between efficiency and throughput. It is likely that performance degradation occurs beyond this point as a result of resource contention and increased overhead in managing higher concurrency levels, which outweighs the benefits of parallel processing.

This test is of critical importance for the purpose of understanding the scalability of the VA, in particular in environments where there is a high volume of certificate status verification requests. Future optimisations may focus on enhancing the system's ability to handle higher loads more efficiently. This may be achieved through better resource allocation, improved software scalability, or hardware upgrades.

5.2.4 Testing SoftHSM performance

In order to gain a comprehensive understanding of the performance of this PKI, it is essential to assess the capabilities of the virtual HSM. For this purpose, a virtual HSM has been temporarily isolated from the EJBCA environment, and the package "OpenSC" has been installed within the HSM container. OpenSC is a set of open-source tools that allows for communication with PKCS#11-compliant devices, in particular, `pkcs11-tool` will be utilised to conduct this test. The program, in fact, allows for the generation and management of keys within a HSM, as well as the signing, verification, and decryption of files through the PKCS#11 API.

Two scripts have been implemented for the assessment, named `sign_softhsm_script.sh` and `dec_softhsm_script.sh`. The first one creates a new SoftHSM token, and afterwards it generates the RSA-4096 keypair using `pkcs11-tool`:

```
pkcs11-tool --module '$softhsm_lib_path' --token-label '$token_label' --login
    --pin '$pin' --keypairgen --key-type rsa:4096 --label '$key_label' --id 01
```

Then, a file `$data_file` must be present within the script directory in order for digital signature to be performed. The script will execute one hundred signatures and will measure performance on a timely basis. This kind of test is effective because the tool does not have a cache, so the same operation can be repeated as though it were the first one.

```
pkcs11-tool --module '$softhsm_lib_path' --token-label '$token_label' --login
    --pin '$pin' --sign --mechanism SHA256-RSA-PKCS --input-file '$data_file'
    --output-file '$signature_file' --id 01
```

At the end, if there no errors occur, the script will print the average time taken for a single signature and the total elapsed time from the first to the last signature. As a result, the average time taken for a single signature of a 826 B file was 42 ms, while the total elapsed time was 4.641 s.

The `dec_softhsm_script.sh` script, as the other one, generates a SoftHSM token and a RSA-4096 keypair. Then, it creates a 256 B `file.txt` with the command `truncate` which is afterwards encrypted with OpenSSL using the generated public key:

```
# Extract the public key
pkcs11-tool --module '$module_path' --token-label '$token_label' --login
    --pin '$pin' --read-object --type pubkey --id 01 --output-file
    'public_key.pem'

# Create a file of size 256 B
truncate -s 256 'file.txt'

# Encrypt the file using OpenSSL and the public key
openssl pkeyutl -encrypt -in 'file.txt' -out 'file_to_decrypt.enc' -pubin
    -inkey 'public_key.pem'
```

Eventually, the script will perform the decryption using using `pkcs11-tool` one hundred times and will print the average time taken for a single decryption and the total elapsed time from the first to the last signature.

```
pkcs11-tool --module '$module_path' --token-label '$token_label' --login
    --pin '$pin' --id 01 --decrypt --mechanism RSA-PKCS --input-file
    'file_to_decrypt.enc' --output-file 'decrypted.txt'
```

The execution of the script resulted in 61 ms of average time for a single decryption and 6.491 s for the entire decrypting process.

The obtained metrics demonstrate that SoftHSM can handle high throughput signing operations, making it suitable for environments where frequent signing is required, and highlight its ability to efficiently handle decryption tasks, even with large key sizes such as RSA-4096. Despite being a software-based solution, SoftHSM delivers robust performance that meets the needs of this project. However, for larger deployments or scenarios requiring higher security guarantees, the integration of hardware HSMs could be considered to further enhance performance and security. These findings provide a solid foundation for PKI performance and guide future optimisations and scalability considerations.

5.3 Security

This section delves into the security considerations of the designed PKI ecosystem, evaluating the strengths and weaknesses inherent in the deployment of EJBCA CE and a virtual HSM, and providing an analysis of the vulnerabilities introduced by the choices of technology and architecture. The discussion will encompass the effectiveness of EJBCA CE in establishing robust certificate management and authentication processes, as well as the security implications of employing a virtual HSM as opposed to traditional physical HSM solutions. Eventually, this thorough examination will culminate in highlighting critical security insights that can guide the optimization of the PKI implementation for enhanced protection and compliance with established security standards.

Before starting the assessment, it is appropriate to consolidate those security features that were introduced and implemented within the designed PKI infrastructure, throughout the previous Chapters. To perform cryptographic operations, EJBCA uses the Bouncy Castle libraries, an open-source project powered by PrimeKey, that provides a lightweight cryptography API for Java and C# [43]. Such libraries allow the framework to support an high variety of algorithms, such as RSA, ECC, and even quantum-ready ciphers for key generation and management, and SHA2 and

SHA3 as hashing algorithms; furthermore they provide support for managing the communication with the HSMs and protocols like OCSP, TLS, CMP, etc.

The EJBCA server, deployed within a Docker Container, is protected by a password that secures it at the operating system level; while the EJBCA operations, for instance accessing to the Admin Web interface, issuing and managing certificates, all require a certificate-based authentication. To further enhance the security posture, the PKI employs a Role-Based Access Control (RBAC), that defines specific roles to adhere to the PoLP (Principle of Least Privilege), and delineates clear operational boundaries for each actor within the system, ensuring that each user or automated entity has only the minimum level of access required to perform its tasks. A strict allocation of permissions, indeed, minimises the risk of security breaches and mitigates the impact of cyber attacks. In the configuration settings, it is also possible to enable the session timeout, i.e. if a user leaves their device unattended while logged in, the session will automatically terminate after a certain period of inactivity, thus preventing potential misuse by unauthorized individuals.

Allowing only few Firewall Ports is another enhancement, in fact in this architecture only the ports required to make EJBCA work without problems stay open. The application needs the port `8443`, which requires TLS client certificate for authentication, to access to the restricted functionalities of EJBCA, like the Admin Web and the enrolment protocols; then the TLS port `8442`, to allow unauthenticated users to access public, yet secured, functionality, such as EJBCA RA Web enrolment; HTTP port `8080` needs to be open to access to public data like AIA or CDP. The database server works with the default port `3306` and, last but not least, if a SCP server is implemented, this may work with port `22`.

5.3.1 Risks and Vulnerabilities

Although the numerous precautions that can be taken, achieving a secure PKI is challenging. The proposed architecture is not considered a top-tier solution for several reasons. The CAs, and particularly root CAs, serve in fact as the cornerstone of any infrastructure because they issue trust at the highest level, and this characteristic makes them are unique in their ecosystem. As the sole issuers of trust for subordinate CAs and other certificates, root CAs represent a Single Point of Failure (SPOF), i.e. any compromise or failure in the root CA will undermine the entire trust model of the PKI. To mitigate this problem, the companies tend to isolate their CAs from the external network, leaving the interaction with the end entities to the RAs. As stated in the previous chapters, EJBCA CE does not permit the separation of the CA with the RA, making this vulnerability even more critical. Even if RBAC that restricts access to sensitive areas, the CAs will remain connected to the external network, which is not considered secure.

According to the database on CVE Details, which contains products' discovered Common Vulnerabilities and Exposures, EJBCA has encountered significant security challenges in the recent years, including Cross-site Request Forgery (CSRF) and Cross-Site Scripting (XSS) [44]. CSRF attacks deceive the browser into executing unwanted actions in a web application where a user is authenticated, potentially leading to unauthorised commands. The second type of attack consists in injecting malicious scripts into content from trusted websites, thus bypassing access controls. Clearly, the EJBCA team is actively engaged in the identification and mitigation of these issues, releasing patches for the application as soon as possible. However, it should be noted that such security updates are not included in the Community Edition. Furthermore, EJBCA CE lacks a Service Level Agreement (SLA), which means there is no guarantee of timely developer support. These aspects expose a company to significant security threats and increase the risks.

Another aspect to consider is the protocol compatibility. While EJBCA CE supports SCEP and CMP, the absence of support for more recent enrolment protocols, such as ACME and EST, can be particularly limiting for environments that demand high levels of automation and scalability in certificate management. The absence of these protocols restricts applicability of the solution in scenarios where automation and high scalability of certificate management are required. SCEP, though functional, does not offer the same level of security and automated certificate renewal and management that ACME provides; similarly, CMP, while more feature-rich than SCEP, still lacks the simplicity and integration ease of ACME and EST for modern environments such as cloud

services and DevOps workflows. This limitation may necessitate additional administrative effort and custom development to bridge the automation gaps, which could be inefficient and prone to errors in large-scale or rapidly evolving technology landscapes.

In the designed PKI there is an important actor, JBoss, that serves as the application server that manages the deployment and operation of Java applications, including the EJBCA software. A significant vulnerability in this configuration is its susceptibility to Denial of Service (DoS) attacks, particularly through the submission of overly large data packages. The JBoss server is unable to restrict the size of data packets it receives via HTTP requests due to the nature of TCP protocol, leaving the system vulnerable to attack strategies that involve sending large volumes of data to overwhelm the server.

Another significant weakness identified in the PKI implementation is the adoption of a virtual HSM. Although SoftHSM utilises reliable OpenSSL libraries, it remains a software solution and cannot match the robustness of a real Hardware Security Module, which is inherently hardware-based. Physical HSMs are considered more secure primarily due to their method of random number generation. Applications, such as OpenSSL, derive seeds from various sources that may include hardware events, operating system data, and other inputs, and this procedure is called Pseudo Random Number Generation (PRNG). Real HSMs are equipped with specific hardware components that generate numbers using more reliable methods based on thermal, avalanche, and atmospheric noises, called True Random Number Generators (TRNG). Additionally, physical HSMs are tamper-resistant, offering visible signs of tampering attempts, whereas software programs are more susceptible to security breaches if system vulnerabilities are exploited.

In the light of these vulnerabilities, adopting an architecture of this kind in a production environment presents substantial risks. The inherent weaknesses associated with the use of virtual HSMs, when combined with the limitations of the EJBCA CE in supporting advanced protocols and robust access management, could have severely detrimental impact on the security and efficiency of a PKI.

5.3.2 Security Best Practices and Recommendations

Having previously identified the intrinsic weaknesses of the proposed PKI, it is now necessary to discuss the ways in which these problems can be mitigated and the security guidelines that a company should follow when implementing a system of this kind.

The first thing to do is to reduce the exposure of the CAs, especially the Root CAs. Typically, Root CAs are only active when renewing Sub CAs certificates or when issuing CRLs, which are sporadic tasks. For this reason, many organisations tend to disable the machine that hosts the Root CA and subsequently protect it within a secure facility. Once this has been done, the machine will be activated once more to perform the necessary operations when they are required and then locked again. In contrast, Sub CAs handle a higher volume of certificates, and therefore they must remain activated to perform certificate management tasks. The optimal approach for these CAs is to segregate them from the external network and utilise automated services that renew certificates and CRLs when possible, and to allow the staff to perform manual operations from the internal network. As stated in the previous chapters, the issuance of certificates is delegated to designated RAs and protocols, while certificate verification is performed by VAs.

Another best practice is disabling the functions that are not required in the ecosystem, in compliance with PoLP. This configuration ensures that only the necessary functionalities and access are exposed, thus reducing the potential attack surface. For instance, in the context of the current project the OCSP protocol has been disabled in the CA module because the OCSP requests are handled by the VA, whereas in the VA the RA function is deactivated along with other non-required issuing functionalities. In the event that the CMP and SCEP protocols are not required, can also be deactivated.

It is important to note that many firewall ports, such as those for HTTP and HTTPS, must remain standard in order to ensure external accessibility. However, for other ports like those used for databases, it is recommended to change them to non-standard, adopting private ones. This

strategy enhances security by reducing the risk of unauthorised access risks to the PKI's sensitive content.

To mitigate the aforementioned problem of the system's susceptibility to DoS attacks due to the submission of very large data packets, it is possible to utilise firewall and proxies. These tools can indeed be configured to restrict the size of incoming data requests, thus providing a barrier against this kind of threat by preventing excessively large packets from reaching the server.

It is also important to protect servers, modules and tokens with robust passwords. These passwords must comprise uppercase and lowercase letters, symbols and numbers, and must not contain any references to organisational information or data. This approach helps to reduce the likelihood of successful dictionary and rainbow table attacks. In the current PKI installation, which is running on Docker containers, the sensitive data is more exposed. For instance, the file `docker-compose.yml` contains the clear text password of the services. Consequently, permissions and ownership policies must be applied to the folder that stores the containers' information and to all its elements.

Finally, it is key to recognise that redundancy is crucial to guarantee a secure system. Although this implementation implies a significant financial outlay, it is important to employ replicas of the components, including not only the PKI itself, but also HSM, firewalls, load balancers etc. For example, having a single HSM is nearly as risky as having none due to the lack of redundancy, because if that sole HSM fails or is compromised, the whole cryptographic operations and key management can be disrupted, leading to potential security breakdowns and system unavailability. Furthermore, it is recommended that two different brands of such modules be adopted, in order to ensure that the PKI environment remains operational even in the event of a vendor firmware malfunction.

5.4 Use cases

EJBCA CE represents a lightweight solution that enables the implementation of a scalable, flexible and affordable PKI in small environments. There are many possible applications [45]:

- Create a test PKI environment to explore its functionalities learn how to configure, manage and monitor its elements.
- Issue TLS and mTLS certificates. EJBCA CE provides a robust PKI solution for issuing TLS and mTLS certificates, suitable for testing and prototyping. This flexibility makes it ideal for secure communications in IT infrastructure, IoT, industrial setups, and DevOps, ensuring that organisations can efficiently manage private trust environments where mutual authentication is required.
- Try quantum-safe cryptography PKI. EJBCA CE supports the Dilithium and Falcon algorithms. Migrating to quantum-safe cryptography requires assessing compatibility, understanding use case requirements, and integrating hybrid certificates while minimising disruption to existing systems. EJBCA provides a platform for experimentation and a smooth transition to quantum-ready security.
- Digital Identities for IoT Products. For increased security and scalability in IoT prototyping, EJBCA PKI is preferred over OpenSSL and self-signed certificates. Establishing mutual trust between devices requires secure digital identities, which can be achieved using PKI and X.509 certificates. While OpenSSL is suitable for development, EJBCA offers a robust setup for production-ready environments, providing a structured way to generate, manage, and validate the certificates required for secure IoT communications and operations.
- Secure Device Identity in Industrial Cybersecurity (IEEE 802.1AR). This is about establishing secure and authenticated communications in industrial environments, where it is critical for devices such as PLCs and sensors to securely connect and communicate over networks using standards such as TLS and X.509 certificates. This security is critical as more devices in modern factories connect directly to each other over IP protocols and integrate

with systems like SCADA and ERP. EJBCA supports this by enabling the issuance and management of certificates, ensuring that devices authenticate and communicate securely from initial deployment through their operational lifecycle.

In summary, the EJBCA CE provides a versatile and cost-effective solution tailored for small to medium-sized environments that require a reliable PKI. Its ability to support a wide range of applications – from setting up test environments to facilitating secure communications in industrial environments – makes it an invaluable tool for organisations with basic needs looking to enhance their cybersecurity infrastructure.

Chapter 6

Conclusion

The objective of this Master's degree thesis project was to implement a fully functional PKI leveraging exclusively open-source frameworks and tools, with due consideration for security and performance standards. This was realised by integrating the EJBCA CE framework with a virtual HSM, both of which were orchestrated within a containerised environment under specified conditions and settings that simulate real-world scenarios. Subsequently, a detailed analysis was conducted to evaluate the system's strengths and weaknesses, thereby providing a comprehensive understanding of its operational capabilities. The questions posed in the introduction – regarding the system's reliability and its viability in actual production environments – are now addressed, drawing on empirical data and theoretical insights gathered during the project.

The implemented design effectively demonstrated significant scalability and flexibility, as containerised deployment facilitates changes and updates within the system without necessitating a substantial reconfiguration, thereby reducing downtime and simplifying maintenance processes. From the perspective of performance, it represents a robust solution that can accommodate the issuance of high loads of certificates with considerable throughput. Moreover, performance could be further optimised if a more sophisticated host machine is employed. The utilisation of more advanced processors and faster memory would result in a notable reduction in latency and an increase in processing speed, enabling the accommodation of even larger loads and more demanding tasks with greater efficiency.

Although this PKI solution presents numerous advantages, it is not an optimal system due to a number of limitations which may impact its applicability in certain environments. In fact, the restrictions of EJBCA CE preclude the utilisation of more sophisticated features available in the commercial version, such as advanced automation capabilities and comprehensive management interfaces. For example, this implementation is limited to the establishment of private trust, rather than public trust, which requires a more robust feature set in order to comply with certain standards. Security considerations serve to delineate the constraints of the system. Firstly, the solution is inherently stand-alone, as the CA cannot be separated from the other authorities. Moreover, the reliance on a virtual HSM, despite its practicality and reduced cost, does not afford the same degree of resilience as a physical one. Lastly, the security architecture of the system is largely dependent on a well-managed RBAC, which, in a complex enterprise environment, is a necessary but not sufficient condition. More exposed domains, indeed, require the adoption of additional layers of security controls and monitoring in order to safeguard against sophisticated threats.

In light of the aforementioned considerations, this project has clearly shown its value as an educational tool, offering in-depth explorations of the configuration and management intricacies that students and professionals can leverage for learning and development. Additionally, it represents an adequate solution for PKI architects to test an environment before going to production. Finally, some small to medium-sized businesses (SMBs) that do not have complex security requirements could integrate this system into their private domains to enhance their security posture. In fact, the combination of this PKI with a well-configured firewall allows the organisation to achieve a level of security that meets their exigencies without the need for more sophisticated

and expensive systems. Indeed, the open-source nature of this PKI allows for the deployment of a robust ecosystem with minimal investment, using tools that offer transparency and community support, which are beneficial for continuous improvement and troubleshooting.

This thesis demonstrates that even with open-source tools and a conservative budget, it is possible to deploy a functional and secure PKI system. As the world moves towards a future that increasingly favours password-less authentication and other advanced security protocols, the foundational work laid down by projects like this one will be crucial. This PKI implementation, while primarily educational and suitable for certain business contexts, also sets the stage for future advancements. It provides a framework for the development of scalable, flexible security solutions that can adapt to emerging threats and changing technological landscapes. As the cybersecurity community continues to innovate, the insights gained here will inform the creation of more sophisticated, resilient infrastructures capable of protecting global digital assets in an ever-evolving threat environment.

Bibliography

[1] D. Jewapatarakul and P. Ueasangkomsate, "Digital transformation: The challenges for manufacturing and service sectors", 2022 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunications Engineering (ECTI DAMT & NCON), 2022, pp. 19–23, DOI 10.1109/ECTIDAMTNCON53731.2022.9720411

[2] B. Trzupek, "PKI is key to securing a post-Covid remote workforce", Computer Fraud & Security, vol. 2020, no. 10, pp. 11–13, DOI 10.1016/S1361-3723(20)30108-1

[3] S. Farrell, "Not Reinventing PKI until We Have Something Better", IEEE Internet Computing, vol. 15, no. 5, 2011, pp. 95–98, DOI 10.1109/MIC.2011.120

[4] P. Gutmann, "PKI: it's not dead, just resting", Computer, vol. 35, no. 8, 2002, pp. 41–49, DOI 10.1109/MC.2002.1023787

[5] Polaris Market Research, "Public Key Infrastructure (PKI) Market Share, Size, Trends, Industry Analysis Report, By Component; By Deployment Type; By End-Use (BFSI, IT & Telecommunication, Government & Defense, Retail & E-commerce, Healthcare, Manufacturing, Others); By Region; Segment Forecast, 2023 - 2032", https://www.polarismarketresearch.com/industry-analysis/public-key-infrastructure-pki-market, 2023

[6] B. Rajendran, "Evolution of pki ecosystem", 2017 International Conference on Public Key Infrastructure and its Applications (PKIA), 2017, pp. 9–10, DOI 10.1109/PKIA.2017.8278951

[7] C. Adams and S. Lloyd, "Understanding PKI: Concepts, Standards, and Deployment Considerations", Addison-Wesley Professional, second ed., November 2002, ISBN: 978-0321743091. Part I

[8] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC-5280, May 2008, DOI 10.17487/RFC5280

[9] D. W. S. Ford, D. S. Chokhani, S. S. Wu, R. V. Sabett, and C. C. R. Merrill, "Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework", RFC-3647, November 2003, DOI 10.17487/RFC3647

[10] International Telecommunication Union – Telecommunication Standardization Sector, "ITU-T Recommendation X.509: Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks", https://www.itu.int/rec/T-REC-X.509-200003-S, March 2000

[11] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and D. C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC-6960, June 2013, DOI 10.17487/RFC6960

[12] P. Gutmann, "Simple Certificate Enrolment Protocol", RFC-8894, September 2020, DOI 10.17487/RFC8894

[13] J. Young and A. Honore, "Simple Certificate Enrollment Protocol Overview", https://www.cisco.com/c/en/us/support/docs/security-vpn/public-key-infrastructure-pki/116167-technote-scep-00.html, August 2016

[14] M. Pritikin, P. E. Yee, and D. Harkins, "Enrollment over Secure Transport", RFC-7030, October 2013, DOI 10.17487/RFC7030

[15] T. Mononen, T. Kause, S. Farrell, and D. C. Adams, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC-4210, September 2005, DOI 10.17487/RFC4210

[16] R. Barnes, J. Hoffman-Andrews, D. McCarney, and J. Kasten, "Automatic Certificate Management Environment (ACME)", RFC 8555, March 2019, DOI 10.17487/RFC8555
[17] M. A. Kushner, "Enlightened enrollment: The book of five certificate enrollment protocols", https://www.keyfactor.com/blog/what-is-acme-protocol-and-how-does-it-work/, March 2022
[18] M. Thompson, "What is ACME protocol and how does it work?", https://www.primekey.com/resources/enlightened-enrollment-the-book-of-five-protocols/, April 2021
[19] Keyfactor, https://www.ejbca.org/
[20] Keyfactor, "EJBCA Community vs Enterprise", https://www.ejbca.org/community-vs-enterprise/
[21] Keyfactor, "EJBCA Concepts", https://doc.primekey.com/ejbca/ejbca-introduction/ejbca-concepts
[22] Nick Naziridis, "Comparing ECDSA vs RSA", https://www.ssl.com/article/comparing-ecdsa-vs-rsa/, June 2018
[23] Y. Genç and E. Afacan, "Design and implementation of an efficient elliptic curve digital signature algorithm (ecdsa)", 2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS), 2021, pp. 1–6, DOI 10.1109/IEMTRONICS52119.2021.9422589
[24] Soatok, "Guidance for Choosing an Elliptic Curve Signature Algorithm in 2022", https://soatok.blog/2022/05/19/guidance-for-choosing-an-elliptic-curve-signature-algorithm-in-2022/, May 2022
[25] Keyfactor, "EJBCA Concepts", https://doc.primekey.com/ejbca/tutorials-and-guides/tutorial-start-out-with-ejbca-docker-container
[26] Tomas Gustavsson, "Integrate your VAs with your CA using Secure copy protocol (SCP)", https://www.ejbca.org/community-vs-enterprise/, January 2023
[27] Keyfactor, "SCP Publisher", https://doc.primekey.com/ejbca/ejbca-operations/ejbca-ca-concept-guide/publishers-overview/scp-publisher
[28] Keyfactor, "Certificate and CRL Reader Service", https://doc.primekey.com/ejbca/ejbca-operations/ejbca-ca-concept-guide/services/certificate-and-crl-reader-service
[29] I. Aciobanitei, L. Leahu, and M. Pura, "A PKCS#11 Driver for Cryptography in the Cloud", 2018 10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), 2018, pp. 1–4, DOI 10.1109/ECAI.2018.8679009
[30] D. Bong and T. Cox, "PKCS #11 Specification Version 3.1", https://docs.oasis-open.org/pkcs11/pkcs11-spec/v3.1/csd01/pkcs11-spec-v3.1-csd01.html, February 2022
[31] S. Mavrovouniotis and M. Ganley, "Hardware Security Modules", pp. 383–405. Springer New York, 2014
[32] OpenDNSSEC, https://www.opendnssec.org/
[33]
[34] Keyfactor, "SoftHSM", https://doc.primekey.com/ejbca/ejbca-integration/hardware-security-modules-hsm/softhsm
[35] Keyfactor, "Security Audit Events", https://doc.primekey.com/ejbca/ejbca-operations/ejbca-ca-concept-guide/logging/audit-log-overview/security-audit-events
[36] Keyfactor, "Monitoring and Healthcheck", https://doc.primekey.com/ejbca/ejbca-operations/ejbca-operations-guide/ca-operations-guide/ejbca-maintenance/monitoring-and-healthcheck
[37] Tildeslash, "Monit", https://mmonit.com/monit/
[38] Keyfactor, "Monitor EJBCA host using Monit", https://doc.primekey.com/ejbca/tutorials-and-guides/monitor-ejbca-host-using-monit
[39] Keyfactor, "What is Certificate Management?", https://www.keyfactor.com/education-center/what-is-certificate-management/
[40] MariaDB Foundation, "MariaDB", https://mariadb.org/
[41] Keyfactor, "Maximizing Performance", https://doc.primekey.com/ejbca/ejbca-installation/maximizing-performance
[42] Keyfactor, "EJBCA Client Toolbox", https://doc.primekey.com/ejbca/ejbca-operations/ejbca-operations-guide/command-line-interfaces/

ejbca-client-toolbox
[43] The Legion of the Bouncy Castle, "Bouncy Castle", https://www.bouncycastle.org/
[44] CVE Details, "Primekey EJBCA – Products, threats and statistics", https://www.cvedetails.com/product/83348/Primekey-Ejbca.html?vendor_id=23203
[45] Keyfactor, "Here is your Certificate use case", https://www.ejbca.org/use-cases/