Technische Universität München

Department of Informatics

Master's Thesis in Informatics

Design and Implementation of a Management Service for Digital Certificates

Pranav Jagdish

Technische Universität München

Department of Informatics

Master's Thesis in Informatics

Design and Implementation of a Management Service for Digital Certificates

Design und Implementierung eines Management-Service für digitale Zertifikate

Author	Pranav Jagdish
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisor	Matthias Wachs
Date	July 14, 2016

Informatik VIII
Chair for Network Architectures and Services

I confirm that this thesis is my own work and I have documented all sources and material used.

Garching b. München, July 14, 2016

Signature

Abstract

With the deep penetration of Information systems in today's world, the need for secure communication and privacy has also seen a significant rise. Established technologies, namely X.509 and OpenPGP that can help in this domain, tend to be out of the reach of the common user and even difficult to use for expert users. This thesis addresses this issue by analysing how an effective, usable and secure system can be implemented for organisations, thus furthering the goal of secure communication. The thesis analyses technologies and systems in use or available today, and analyses the various aspects that make secure communication difficult and challenging. Using this analysis, the thesis presents the design and resultant implementation for a Management Service for Digital Certificates which covers the whole certificate lifecycle and supports both the established technologies i.e. X.509 and OpenPGP. The thesis along with the resultant system, will provide valuable help to organisations – both big and small – in securing their communications while simultaneously making it easy and less challenging for a common user.

Zusammenfassung

Mit der weiten Verbreitung von Informationssystemen in der heutigen Zeit steigt gleichzeitig auch die Notwendigkeit der sicheren und die Privatheit der Nutzer schützenden Kommunikation. Etablierte Technologien wie X.509 und OpenPGP, die auf diesem Gebiet helfen können, sind für die Verwendung durch normale Nutzer wenig geeignet und auch für technisch versierte Nutzer schwierig zu verwenden. Diese Arbeit beschäftigt sich mit dieser Problematik indem sie analysiert, wie ein effektives, verwendbares und sicheres System für Organisationen realisiert werden kann, das die Sicherheit in der Kommunikation erhöht. Diese Arbeit analysiert Technologien und Systeme, die heute in Verwendung oder verfügbar sind und unterschiedliche Aspekte, die Kommunikation schwieriger und herausfordernder machen. Basierend auf dieser Analyse präsentiert diese Arbeit das Design und als Ergebnis die Implementierung eines Dienstes zur Verwaltung digitaler Zertifikate, das den gesamten Lebenszyklus digitaler Zertifikate abdeckt und die beiden etablierten Technologien X.509 und OpenPGP unterstützt. Diese Arbeit bietet zusammen mit dem resultierenden System eine wertvolle Hilfe für sowohl große und kleine Organisationen, die ihre Kommunikation absichern wollen und dies gleichzeitig einfach und intuitiv für Nutzer realisieren wollen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since time immemorial, one of the most important social tasks that humans undertake - is to communicate. With time, modes of communication have changed. In today's world, the optimal method of communication is the E-Mail.

E-Mail, itself, has come a long way since the first ARPANET E-mail was sent in 1971 [1]. Its popularity increased massively as the personal computer and Internet started penetrating the consumer market. Organizations - both big and small - as well as home users started using the E-Mail as their preferred mode of communication. The advantages of email - from the speed to just the ease of writing one - over traditional mail, only helped increase its popularity.

With its rise in popularity, various security problems [2] also came to the fore. As hackers of all colours began dissecting the email - various loopholes became known. Soon, hackers started exploiting these loopholes for personal gains. As more and more organizations started deploying E-Mail services and started moving away from traditional systems - more and more cases of email related fraud started happening.

Computer researchers as well as the industry began deploying countermeasures to correct the flaws in E-Mail systems. Many of the flaws were closely associated with the underlying technologies and protocols of E-Mail. Owing to this, and the continued use of the same protocols and technologies, E-Mails are still susceptible to a host of security problems and issues. Even the countermeasures that were released by researchers or the industry were a source of controversy. For example when PGP was released in 1991 [3] [4], it was perhaps the first complete email security package. However, controversies soon followed its release. As it was freely available over the Internet, the US Government had claimed that since foreigners can obtain it, its a violation of laws concerning the export of munitions. To get around this restriction, later versions of PGP were produced outside USA.

Many more new technologies and protocols were created to secure E-Mail. Another

example is Secure/Multipurpose Internet Mail Extensions or S/MIME. This too gained popularity with many users. However, like in the case of PGP, the use of S/MIME and other technologies continued to be limited. Email security, was a complex task for many a normal users as well as even expert users [5]. As computers and other forms of electronic devices were becoming a common place around the world, Governments irrespective of their ideologies or political leanings, started seeing a need to have some form of control over these communications as they did on other forms of traditional communication [6] [7] [8]. International crime and terrorism led governments to initiate clandestine operations of monitoring modern day electronic communication.

In 2013, Edward Snowden, a former Central Intelligence Agency (CIA) employee and a former contractor for the US Government started leaking classified information from the United States National Security Agency (NSA) to the press [9]. Amongst this leaks, were revelations about a surveillance program known as PRISM. The PRISM system collects the e-mail, voice, text and video chats of foreigners and an unknown number of US Citizens from Microsoft, Google, Yahoo, Apple and other technology giants [10].

Following this and other related revelations, Secure E-Mail communication, once again came back in global limelight [11]. However, as before, problems that have persisted with securing E-Mails still remained. The need for security with usability is utmost, and yet is hardly fulfilled.

At the moment, secure communication is a rather difficult task for non-technical users [11] [12]. For an effective secure E-Mail solution, users generally need certificates to progress. A certificate itself has a life cycle - enrolment, distribution, validation, revocation, renewal, destruction and auditing. To understand and to carry out and manage these tasks, a lot of technical knowledge is required. Hence, there is the need for an automated certificate management service that manages the certificates throughout their life-cycle and in turn provides the users with a friendly and easy to use solution to secure their E-Mail communications.

1.1 Goals of the Thesis

The goal of the thesis is to analyse how an effective, usable and secure system can be implemented for big organisations so as to further the goal of secure communication. The thesis elaborates on the design and realization of a management service for digital certificates tailored to the needs of larger organizations and the requirements to enable secure communication. The thesis will research aspects of a decentralized key management service which generate, backs-up, synchronizes and recovers keys - all while adhering to the organizational requirement for key escrow mechanisms and focusing on the users privacy.

1.2 Methodology

A comprehensive approach will be deployed that initiates by trying to look at current technologies, protocols and solutions available for certificate management. State-of-the-art cryptographic approaches will be looked into so as to integrate them suitably with the certification service. This will require familiarization with protocols and standards being used in the real world especially by big organisations. S/MIME (Secure/Multipurpose Internet Mail Extensions) and OpenPGP (Pretty Good Privacy) are to be the foundation blocks of the thesis.

This thesis analyses the life cycle of digital certificates and evaluates the functional and organizational requirements and required components to allow large organizations to issue and manage digital certificates. This will not only require to analyse the technical aspects of certificates like their generation, renewal, revocation, etc. but also the management processes involved in segregating roles and responsibilities so as to protect the certificates from unwanted disclosure.

Based on the analysis of existing technologies, the system will be designed. The design has to be modular and extensible. This will make it possible to adapt the system to various environments as witnessed in the industry and allow it to be easily updated with future requirements. The system will also support the requirement for different roles in a multi-tenant architecture. It has to also fulfil organizational requirements that were elaborated upon in the analysis. It has to be ensured that the system thus designed employs established and state of the art technologies to support organizations in handling the certificate life cycle. Secure processes will also have to be designed to suitably integrate with the certification process. Once a suitable and exhaustive design has been laid out, the system can be implemented.

Evaluation of the resulting system can then be done by comparing the system with similar systems in use today.

1.3 Outline

The thesis is structured as follows: Chapter 2 provides the necessary background information about certificates and other related protocols and technologies. These technologies are the foundations for the thesis. Chapter 3 analyses the problem the thesis wishes to solve, further in detail. Chapter 4 discusses the related work to this thesis and looks at its limitations. Chapter 5 builds upon Chapter 3 and 4 and provides a detailed analysis of the requirements for a certificate management system that will best solve the problem the thesis is addressing. After the complete analysis of our requirements, the certificate management system is designed in Chapter 6, implemented in Chapter

7 and evaluated in Chapter 8. Finally, the thesis concludes in Chapter 9. This chapter also discusses future research topics related to this thesis.

Chapter 2

Background

This chapter discusses various technologies and protocols commonly used to secure email. Without the knowledge of these protocols and standards, it is difficult to progress further. These technologies will be the building blocks for the thesis. In the implementation phase, these technologies will be used to realise the goals of this thesis and hence background information about them is much warranted before proceeding ahead.

2.1 X.509 Certificates and Certificate Signing Requests (CSR)

X.509 is an important standard for Public Key Infrastructure. It is used to manage digital certificates and for public key encryption.

X.509 defines a framework to provide authentication services from the X.500 directory to its users. This directory could be a repository for certificates. Each of this certificate contains a public key and is signed by the private key of a trusted certification authority. X.509 also defines alternative protocols for authentication based on the use of public key certificates. X.509 is an important standard because the certificate structure and the authentication protocols that are defined in it are nowadays used in various contexts [13].

The most prominent part of the entire standard from the point of view of this thesis, are the X.509 Certificates used in S/MIME. In fact, when using the term X.509 certificate, it is actually the IETF's PKIX certificate and CRL Profile of the X.509 v3 certificate standard which is also commonly called PKIX for Public Key Infrastructure (X.509). This is defined in RFC 5280 [14]. This will be discussed in the next section. These certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory either by the CA itself or the user. The certification authority is an entity that issues a certificate binding a public key to a particular distinguished

name (the fully qualified domain name that the certificate is for). X.509 certificates have a hierarchical structure.

The applicant first generates a key pair out of which the private key needs to be kept secret by the applicant. Then the applicant needs to send a Certificate Signing Request to the CA. As the name suggests this is a simple request sent out by an applicant to the CA so as to apply for a digital certificate. This request (the CSR) contains information about the applicant (like the distinguished name) which has to be signed using the private key of the applicant. It also has the public key of the applicant. The CSR process could be (and in most cases is) accompanied by a manual process of checking a user's identity. The typical Information inside a CSR can be seen in the table 2.1.

CSR Field	Description
Distinguished Name	This a fully qualified domain name that you wish to secure. For example: www.abc.com or mail.abc.com. This as you can see also includes the Common Name (www or mail). In case of securing emails, this is the field that can be used to add the name of the user to whom the certificate belongs.
Business name (Or) Organisation name	Usually this is the legal incorporated name of a company and should be included as Ltd, Inc or Corp.
Department Name (Or) Organisational Unit Name	For example Munich, Berlin, London
Town (Or) City	For example Munich, Berlin, London
Province (Or) Region (Or) State	An example could be Bavaria, Florida, etc.
Country	This is the two letter ISO Code for the country where the organisation or individual is based. For example: Germany is DE, India is IN, and so on.
Email address	The email address where the organization can be contacted. This same field is important when it comes to securing emails using X.509 certificates and is used to add the users email address that will be used for purposes of secure email communications.

Table 2.1: Information Inside a CSR

A X.509 certificate is a collection of a standard set of fields containing information about a user or device. Apart from this it also contains the corresponding public key. The X.509 standard lays down what information is present in the certificate and how it is encoded. A X.509 certificate's structure can be seen in the figure 2.1. The fields are explained below.

1. Version - Version of the certificate format. The newest version for X.509 certificates is version 3.

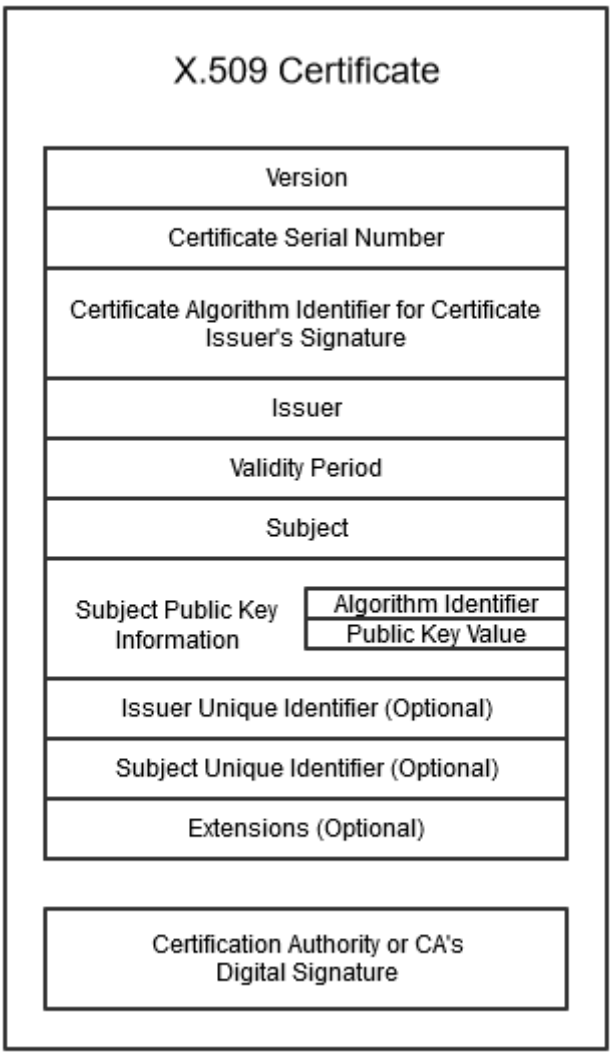

Figure 2.1: Structure of a X.509 Certificate

2. Certificate Serial Number - This is a unique Serial number that the CA assigns to the certificate. The CA has a history of all certificates and it can trace a certificate that it issued using the serial numbers of the certificates. The same applies also to revoked certificates.

3. Certificate Algorithm Identifier - The public key cryptography and message digest algorithms that were used by the issuing CA while digitally signing the certificate.

4. Issuer - The CA that issued the certificate.

5. Validity Period - This is certificates start and expiry dates. The certificate is only valid for this period of time. Note that certificate validity can expire before this

time period in case it's revoked.

6. Subject - The name of the subject or owner of the certificate.
7. Subject Public Key Information - This field has the public key and a list of the public key cryptographic algorithms. These algorithms are for the tasks that the public key can be used for like digital signing, secret key encryption, and authentication.
8. Issuer Unique Identifier - This is an optional information field that can help uniquely identify the issuer of the certificate.
9. Subject Unique Identifier - This is an optional information field that can help uniquely identify the subject of the certificate.
10. Extensions - Additional information can be specified for optional use by PKIs.
11. CA's Digital Signature - This is the digital signature of the CA.

2.2 Public Key Infrastructure X.509 (PKIX)

The Internet Security Glossary in RFC 2822 [15] defines a public key infrastructure as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography. PKI is centre-piece in enabling a secure and convenient way in which public keys can be acquired efficiently [13].

The Internet Engineering Task Force or IETF's Public Key Infrastructure X.509 or PKIX working group has set up a formal model for PKIs based on the X.509 certificates. This model is shown in the figure 2.2.

The following are the key elements of a PKIX Model:

1. End Entity – This is a generic term that can be used to denote end users, devices like servers and routers, or any other entity that can be identified with the subject field of a public key certificate. These entities are the ones that use or support the PKI-related services.
2. Certificate Authority (CA) – This is the entity that issues certificates. It also has a certificate revocation list or CRL. A CA can have many administrative functions but in most cases these are delegated to one or more registration authorities. In brief, the task of the CA is to digitally sign and publish the public key bound to a given end user. This is done by using the CA's own private key. Since, the CA's key can be validated and trusted, the user's key can also be trusted.

3. Registration Authority (RA) – This is an optional element of the public key infrastructure. It can assume many administrative functions from the CA.
4. CRL issuer – Usually the CA has its own CRL but optionally the CA can delegate this task to publish Certificate Revocation Lists (CRL) to some other entity.
5. Repository – This could be a database or directory service that is used to store the certificates and CRLs so that can be retrieved by end entities.

2.2.1 PKIX Management Functions

PKIX identifies many management functions that need to be ported to management protocols. The functions include the following:

1. Registration – By this process an end user makes himself or herself known to the CA. This could be either a direct process or an indirect process in which case it's routed through the RA. Only once the registration is complete will a CA issue a certificate or certificates to the user. Registration basically begins the process of enrolling into a PKI.
2. Initialization – Before a client system can start working properly and securely, it is important that key materials, which have the appropriate relationship with keys stored in other places of the infrastructure, have been installed.

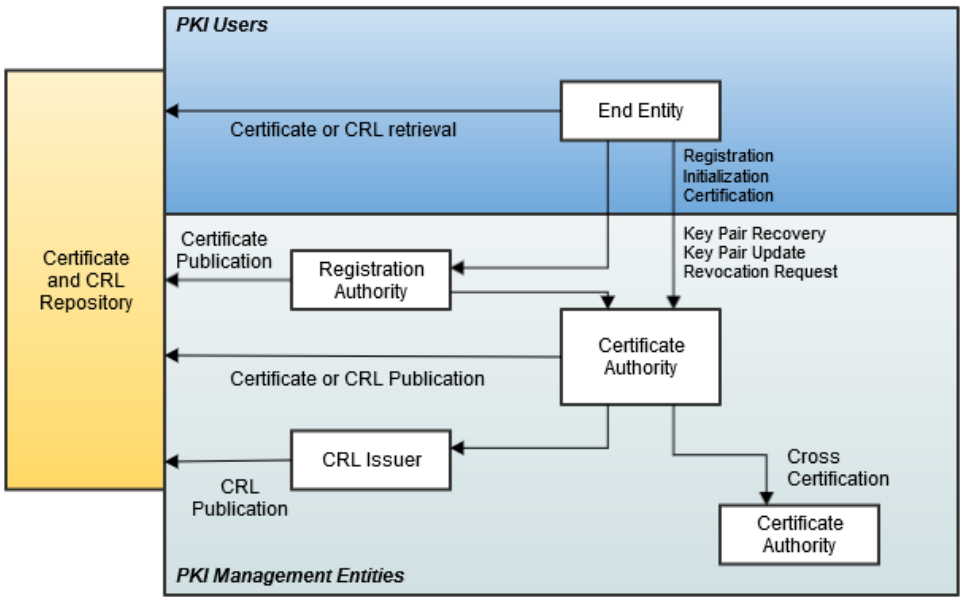

Figure 2.2: The PKIX Architecture Model.

3. Certification – In this the CA issues a certificate for a user using their public key. The certificate is then returned to the user. Optionally this certificate can also be stored in a repository.

4. Key Pair Recovery – Since, keys can get lost, there should be a way to recover the said keys. This is a positive functionality from the end users point of view. There could be various reasons for an end user to lose his or her private keys. This is where the CA can be of help.

5. Key Pair Update – Certificates usually expire after a set term. Hence it is important that all key pairs can be updated regularly and new certificates can be issued whence the key pairs are updated. Update might also be needed in the case of a certificate being revoked.

6. Revocation request – A certificate's private key might get compromised, in which case the certificate needs to be revoked by the CA. There could be other reasons for revocation too like change in a person's name. In this case, this function provides for allowing an authorized person to inform the CA about revoking a certain certificate.

7. Cross Certification – A cross certificate is a certificate issued by a CA to another CA which includes a CA signature key that can be used for issuing certificates. By this function, two CAs can exchange information in creating this cross certificate.

2.2.2 PKIX Management Protocols

The PKIX working group has defined two alternative management protocols that can be used between the various entities in a PKIX model. All the management functions listed in the last section, are supported by these two protocols.

RFC 4210 [16] defines the certificate management protocols or CMP. CMP is an internet protocol that is used for obtaining X.509 digital certificates in a public key infrastructure. In CMP, all of the management functions from the previous section are explicitly identified by specific protocol exchanges. CMP is a flexible protocol.

RFC 2797 [17] defines the certificate management messages over CMS (CMC). Here CMS refers to cryptographic message syntax (RFC 2630 [18]).

2.3 S/MIME

S/MIME stands for Secure/Multipurpose Internet Mail Extensions. It is a standard for signing MIME data as well as for public key encryption [19]. RFCs 3369 [20], 3370 [21], 3850 [22] and 3851 [23] are some of the more important RFCs that define

this standard. S/MIME was originally developed by RSA Data Security Inc. However, now IETF handles the change control of S/MIME. The specification itself is now layered on IETFs' Cryptographic Message Syntax. This specification is very identical to the original PKCS#7 specification that RSA Data Security Inc. had originally developed S/MIME with. Although, S/MIME is on an IETF standards track, S/MIME functionality is built into all modern email software. It is one of the most widely accepted ways to help secure E-Mail along with OpenPGP.

Before S/MIME, system administrators were dependent either on the Simple Mail Transfer Protocol (SMTP) or other proprietary solutions. The SMTP protocol was inherently insecure, while the proprietary solutions had varying degrees of security to them. Many a times, security came at a cost to connectivity. S/MIME helped change this situation by allowing administrators to not compromise either security or connectivity. In this way, S/MIME helped redefine email communication and is probably as important a standard as SMTP itself in the history of email.

S/MIME can provide a number of functions to further the goal of secure email. It provides authentication, message integrity, and nonrepudiation of origin of the email, privacy and data security through encryption.

2.3.1 Functions

By large, S/MIME provides two security services:

1. Digital Signatures
2. Message Encryption

These two services form the basis of email security when it comes to S/MIME. All other concepts related to email security can be seen through these two services [24].

2.3.1.1 Digital Signatures

Digital signatures can be considered to be the digital version of traditional signatures that are used for example to sign contracts or bank cheques. Digital Signatures, by design, provide the same features that traditional signatures provide. These features namely are – authentication, nonrepudiation, and data integrity. It has to be noted though that digital signatures are far more secure than their traditional counterparts. For example, a traditional signature can be counterfeited while its far more difficult to do the same with digital signatures.

Authentication is easy using digital signatures. A signature can easily validate an identity. Just like a clerk at a bank can cross check a signature to verify a person's

identity, so can a digital signature. The digital signature lets the receiver of a message know that it was the sender and not someone else who sent the message.

Digital signatures further ensure that an owner of the signature cannot deny owning the signature. This is what is called nonrepudiation. If a person Bob sends out an email to another person Tom. Tom knows that this email has been sent by Bob (authenticating using the signature) and because of the presence of the signature, Bob cannot deny he did not send the email.

Data Integrity is also a function provided for by digital signatures, with the receiver of a signed email resting assured that the message was not tampered or altered on transit from the sender to the receiver. This is because, if the message would be altered in transit, the signature would get invalidated. Once the receiver knows this, he or she can know for sure that the message has been tampered with.

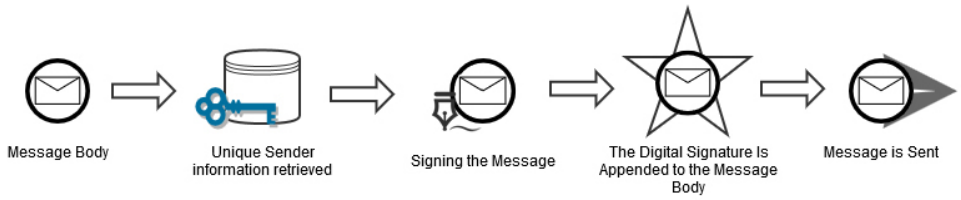

Figure 2.3: The process of digitally signing a message

In Figure 2.3, the process of signing a message can be seen. A message is signed using some unique information of the sender. This unique information can only be provided by the sender. This "digital signature" is appended to the message before the message is sent. It has to be noted, that a malicious user might obtain the unique information that the sender has and could try to impersonate the sender. However, S/MIME can handle such situations and unauthorized signatures are shown to be invalid.

In Figure 2.4, the process of how a signature is verified on the receivers end is seen. Upon receiving the message, the message body is separated from the digital signature. Then the sender's unique information is retrieved from which he or she would have originally signed the message. This information is used to create a digital signature locally. This signature is compared to the signature in the message to complete the verification of the digital signature.

2.3.1.2 Message Encryption

Digital Signatures do not solve the problem of message disclosure. Even if an email is signed, its data is transmitted over plaintext. Any person who can get this email or view

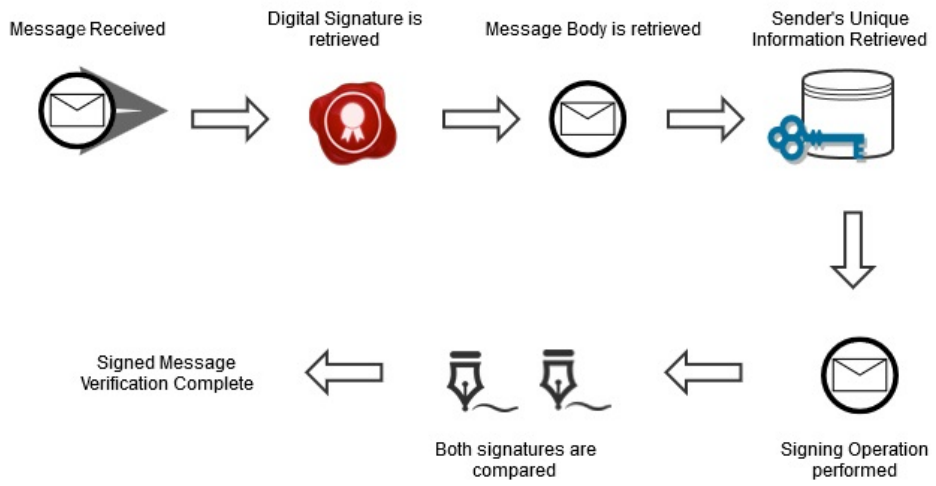

Figure 2.4: Verifying the Digital Signature on a recieved message

it while in transit – legally or illegally – can read the contents of the email. Message Encryption enables two features – confidentiality and data integrity.

As the message is encrypted, the receiver and the sender both can rest assured about the confidentiality of the message being maintained. They can rest assured, that if the message somehow does fall into the hands of individuals or organisations that should not possess it, the contents of the message will not be visible to them even then. As long as the private key of the receiver is safe, the message cannot be viewed by anyone except the receiver (and of course the sender).

Data Integrity is also enabled by message encryption, as an encrypted message cannot be tampered with. If tampered with, it will fail to decrypt at the receivers end thus notifying the receiver that somebody has tampered with the message. This also shields the data in the message from any kind of man in the middle attacks and eavesdropping.

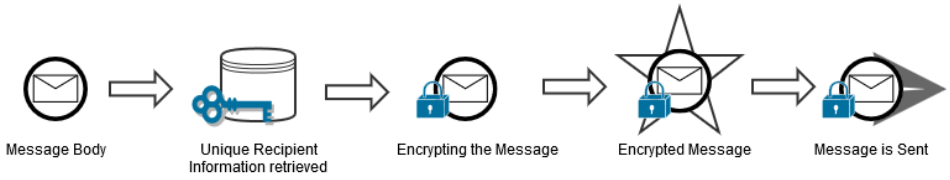

Figure 2.5: Encrypting a Message

In Figure 2.5, the process of how message is encrypted is shown. The recipients unique information (public key for example) is used to encrypt the message, and the encrypted message is then sent to the destination.

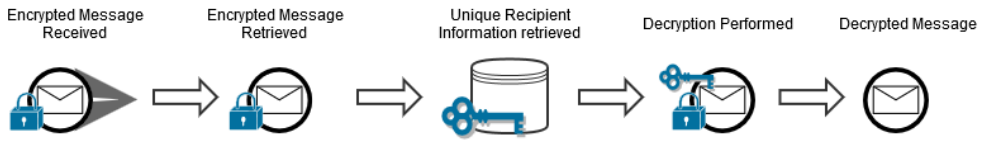

Figure 2.6: Decrypting a Message

Figure 2.6, shows how an encrypted message is decrypted on the receivers end. The recipient requires to have his or her unique information to decrypt the message. For example, their private key.

2.3.2 Certificates

Before, S/MIME can be used for digitally signing emails, encrypting emails or both, a certificate is needed. This certificate has to be issued by a certifying authority or CA. A CA can be an in-house CA (catering to the needs of just the organisation in which it's based) or it could be a public CA (caters to the needs of anyone who so wishes to use its services).

As far as message encryption is concerned, even after getting the certificate issued by the CA, the sending user will need the certificate of the receiving user that he or she wishes to send an email too. Nowadays, many email clients do this task automatically. Once the destination side's certificate is available, the sending side can encrypt an email.

The CA could have varying policies, and dependent on them, the issued certificates (and all their content) maybe publicly posted for reference and verification by other users. Thus, users can lookup other users using their email addresses or names and get their associated certificates. The CAs also maintain a certificate revocation list or CRL, which lists out all the certificates which have been revoked. A revoked certificate implies that any entity that in the future tries to present this certificate should not be trusted.

2.4 OpenPGP

OpenPGP is an industry standard containing protocol specifications for Pretty Good Privacy. Pretty Good Privacy or PGP [3] is an open source email security package. Its a commercial product that follows the OpenPGP standard. It provides for privacy, authentication, digital signatures, and compression. PGP is largely based on existing cryptographic algorithms and does not create any of its own. It's based on RSA, IDEA, and MD5 [24]. PGP and similar software follow the OpenPGP standard, defined in RFC

4880 [25], for encrypting and decrypting data. Another tool like the PGP is the GPG which is an open source software implementation of the OpenPGP standard.

2.4.1 OpenPGP in Operation

OpenPGP can be called a hybrid cryptosystem as it combines some of the best features of both conventional as well as public key cryptography. This hybrid nature of OpenPGP is helpful. Conventional forms of cryptographic encryption is way faster than public key encryption. On the other hand, the public key encryption solves the problem of key distribution. Thus by using both systems together in the OpenPGP hybrid cryptosystem – performance and key distribution are highly improved without compromising security in any way [26].

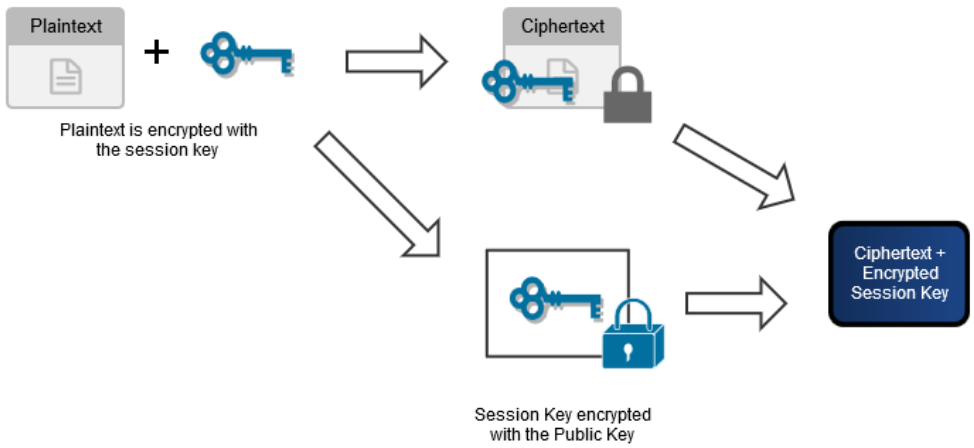

Figure 2.7: Encryption in OpenPGP

2.4.1.1 Encryption

Given some plaintext, OpenPGP first compresses it. This helps in increasing the resistance against cryptanalysis techniques that wish to find out patterns in plaintext and exploit them so as to crack the encrypted data. Compression also helps save disk space as well as the time it takes to transmit the data.

After this OpenPGP creates a session key which is much like a secret key that can be only used once. This key is generated using a random number generated from the random movements of the mouse and typed keystrokes. This random user input from the mouse and keyboard is only to add extra randomness to the system.

This session key then works with a secure cryptographic algorithm and encrypts the plaintext. Once the encryption is complete, the session key is encrypted with the receiver's public key. Then, the encrypted data is sent along with this session key to the receiver.

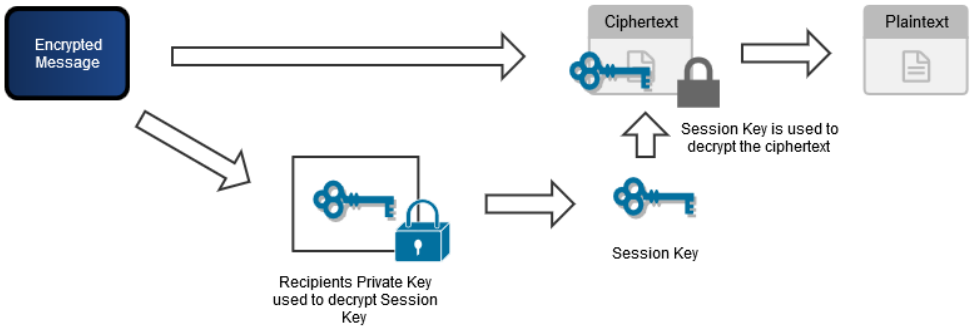

Figure 2.8: Decryption in OpenPGP

2.4.1.2 Decryption

This is exactly the reverse of the encryption process explained above. OpenPGP on the receiver's side, uses the private key of the receiver to first decrypt the session key that has been sent along with the encrypted data. Once the session key has been decrypted, OpenPGP can use it decrypt the encrypted data.

2.4.1.3 Key Storage

OpenPGP stores the keys in two different files on disk. One is for public keys, while the other is for private keys. These files are called keyrings in OpenPGP terminology. Public keys of recipients can be added to the public keyring. Keeping the private keyring safe is very important. If this keyring is lost, a user will not be able to decrypt any OpenPGP encrypted messages.

2.4.2 Digital Signatures

The system of digital signatures as discussed earlier had a few problems. Firstly, it is slow and produces a huge volume of data. This could be almost double the size of the original information we wished to sign. This can be improved by using a one way hash function. What this hash function does is, it takes a variable length input (in the case of signing emails this could be a message of any length), and it produces a fixed output. Basically, no matter how long an input is provided to this hash function, it will always

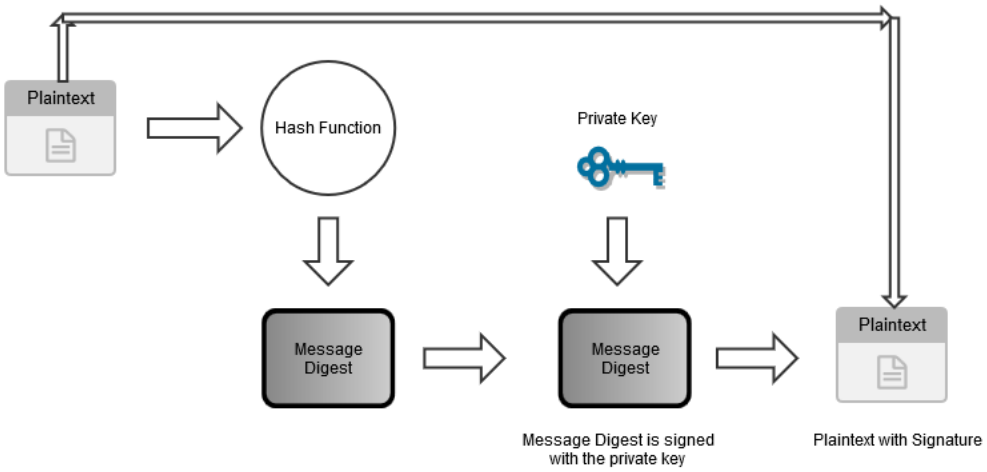

Figure 2.9: Using Digital Signatures in OpenPGP

give us a same fixed length of output. While the hash function makes sure the output length is always fixed it also ensures that any changes to the content being passed into it, will definitely affect the final output produced.

OpenPGP uses a similarly strong hash function on the plaintext that a user is signing. This hash function generates a message digest. A message digest is a fixed length data item. OpenPGP uses this digest and the private key to create the digital signature which can be transmitted alongside the message. On the receiving end, the recipient can re-compute the message digest to ensure that the message was not tampered with thus verifying the signature. Even a slight change to the actual message will compute a different hash that will not match the digital signature. In OpenPGP, digital signatures play a very important role in authenticating and validating another users OpenPGP keys. The process of digitally signing in OpenPGP is shown in Figure 2.9

2.4.3 Digital Certificates

Digital certificates have already been discussed in the preceding sections. They are also used in OpenPGP. OpenPGP recognises two formats of certificates – OpenPGP certificates and X.509 certificates. X.509 certificates have already been discussed in detail. Here, OpenPGP certificates will be further elaborated upon.

OpenPGP certificates are unique in one aspect that a single certificate may contain multiple signatures. Furthermore, these signatures might only authenticate some labels and not all labels present in the certificate. Different people may have different opinions about how to check for authenticity, and therefore what one person may authenticate with a signature might be different from another person. A signature attests to the

Version Number	This identifies which version of OpenPGP was used to create the key associated with this certificate.
Public Key	This is the public key of the certificate holder. It also has the algorithm used to create this key - either RSA, Diffie-Hellman or Digital Signature Algorithm (DSA).
Certificate Holders Information	This consists of the identity information about the user, such as the user's name, their ID, and so on.
Digital signature	This is called a self-signature. This signature is made using the private key that corresponds to the public key in this certificate.
Validity period	This is the time period for which the certificate is valid for. It has a start date and an expiration date.
Key Encryption Algorithm	This is the symmetric encryption algorithm which the certificate owner prefers to use to encrypt information. This could be either CAST, IDEA or Triple-DES.

Table 2.2: Information Inside a OpenPGP Certificate

authenticity that a label is associated with a particular public key. However, in no way does a signature attest to the authenticity of all the labels associated with that public key. OpenPGP certificates differ from X.509 certificates in many ways. Some of the differences are as follows. OpenPGP certificates can be created by a user themselves whereas a X.509 certificate has to be issued by a certifying authority. In the case of OpenPGP certificates anyone can play the role of a validator, whereas for X.509 certificates, only the CA can validate a certificate. X.509 certificates support only a single name for the key's user. Furthermore, X.509 certificates support only a single digital signature to attest to the keys validity whereas OpenPGP certificates can have numerous signatures for validating a key.

2.4.3.1 Web of Trust

Before understanding the OpenPGP web of trust, its important to understand two important OpenPGP specific terms - meta-introducers and trusted introducers. A meta-introducer is very much similar to a CA in a PKIX environment. It is the entity most trusted by all users just like a CA. A meta-introducer, typically delegates the ability to trust keys upon others who then become known as trusted introducers. These trusted introducers can validate keys to the same effect as can the meta-introducer. The key difference between a meta-introducer and a trusted introducer is that, only a meta introducer can create more trusted introducers. A trusted introducer cannot create more trusted introducers. In a PKIX environment, the meta-introducers are similar to the Root CA, while the trusted introducers are similar to subordinate CAs.

In a typical PKIX environment, most of the times people rely on a CA to establish a

certificate's validity. In brief, the users trust the CA. In OpenPGP's view of trust, the case is very different. Key authentication in OpenPGP depends on a distributed trust model called the web of trust. This model uses a decentralized system of trusted introducers. Any user can be a trusted introducer and thus act as a CA. OpenPGP allows anyone to sign anyone else's public key certificate which ultimately creates a web like structure of validations.

Any OpenPGP user can validate another OpenPGP user's public key certificate. However, such a certificate is only valid to another user if he or she recognize the user validating the certificate as a trusted introducer.

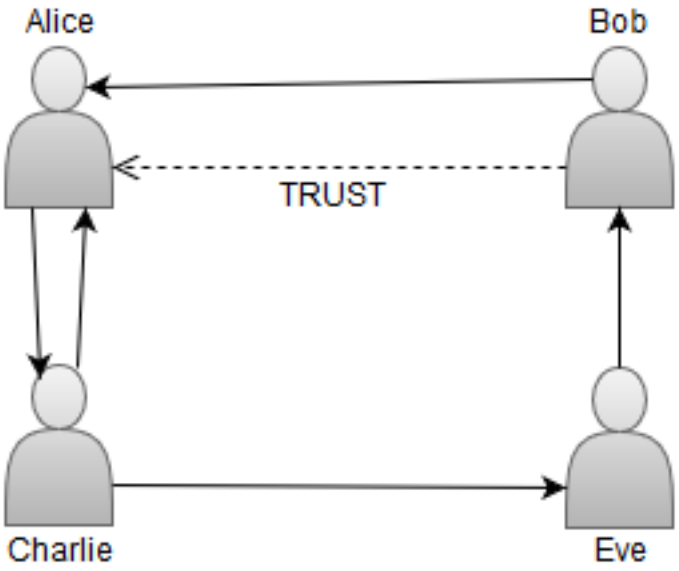

Figure 2.10: An example for Web Of Trust

In Figure 2.10 ,you can see an example for a web of trust. An arrow from Bob to Alice means that Bob has signed a key of Alice. A dotted arrow between them means that Bob trusts Alice to be an introducer. Now, since Alice signed Charlie's key, Alice becomes a trusted introducer for Bob to Charlie. Hence, Bob trusts Charlie's key to be the real one.

2.4.3.2 Revocation of OpenPGP Certificates

The web of trust that OpenPGP uses, creates an interesting scenario when it comes to revocation of certificates. In PKIX environemnts, as the CA was the only one to sign the X.509 certificate, it can revoke its signature from the certificate.

However, in an OpenPGP environment, multiple signatures exist in a certificate. In this case, anyone who has signed a certificate can revoke his or her signature on the certificate, To do this the person will have to use the same private key with which he or

she created that signature in the first place. When a signature is revoked, this indicates to other users that this particular user (who has revoked the signature) does not trust the certificate anymore. A simple reason for this could be that the user believes that the private key of the certificate has been compromised or the identity data that corresponds to the certificate is no longer correct. In case of OpenPGP, a revoked signature carries as much significance as a revoked certificate. If the whole certificate needs to be revoked, then this can only be done by the owner of the OpenPGP certificate i.e. the user who holds the corresponding private key of the certificate. The owner of the OpenPGP certificate can also designate a revoker who can revoke the certificate on his or her behalf when the need be.

Informing other users about the revocation of a certificate is also different in the case of OpenPGP environments. As far as PKIX environments are concerned, as discussed before, a CA maintains a certificate revocation list which is basically a list of all revoked certificates. In case of OpenPGP certificates, the most common way to communicate to other users that a certificate has been revoked is to post it on a certificate server (synonymously known as a key server).

Chapter 3

Analysis

This chapter analyses the problem that this thesis is trying to solve. It further analyses each step of the certificate lifecycle. Going through each step of the certificate lifecycle and analysing it is important. Owing to this analysis, the requirements for a certificate management service can be laid down.

3.1 Problem Analysis

As stated earlier in the Introduction chapter, the goal of the thesis is to analyse how an effective, usable and secure system can be implemented for big organisations so as to further the goal of secure communication. The thesis elaborates on the design and realization of a management service for digital certificates tailored to the needs of larger organizations and the requirements to enable secure communication. The thesis will also research aspects of a decentralized key management service which generates, backs-up, synchronizes and recovers keys - all while adhering to the organizational requirement for key escrow mechanisms and focusing on the users privacy.

From this we can understand the problem the thesis wishes to address. First and foremost, the problem of security and usability. The certificate management service that is to be designed and implemented, needs to be usable. Unlike, other available services discussed before, It needs to be easy to grasp and easy to use for a normal user while not compromising on security. Secure communication continues to remain a hard task for many, and hence the certificate management service wants to make it an easy procedure. By using state of the art secure cryptographic algorithms, the security aspect of the new service can be maintained.

Furthermore, owing to the fact that large organizations may decide to use either S/MIME or OpenPGP or even both – compatibility for both these technologies needs to be provided by the new service.

Providing the function of backing up keys and recovering them, is an additional and very helpful feature for the end users. There could be numerous reasons for a user to lose his or her private keys. This option helps nullify the effects of such a problem.

Before the entire problem can be analysed to provide us with an overview of the requirements of the new service, a more detailed look into various aspects of the centre-piece of the service – certificates – is needed. By looking at each step in the lifecycle of a certificate a better understanding of what functions a typical certificate management service needs to perform can be reached. This analysis is done in the next section.

3.2 A General Certificate Lifecycle

The following sections look into the general certificate lifecycle and what happens in each step of a certificates life.

3.2.0.1 Enrolment

This step is more of a cooperative task between the CA and the user. The users requesting a certificate from the CA initiate the request. This enrolment request generally contains the public key of the user and the enrolment information. Once the user requests a certificate from the CA, the CA verifies the information sent to it on the basis of its own policies. If the policies stand with the certificate, it will issue the certificate, post it to a repository for example, and send back to the user an identity certificate that he or she can now use.

This step can be broken down into the following sub steps from the explanation above:

1. Here the user registers with the CA, either directly or through a RA, thus making himself or herself known to the CA. Only once the registration is complete, will a CA issue a certificate or certificates to the user.

2. The User now creates a request for a certificate and sends it to the CA. This is simple Certificate Signing Request when talking about X.509 certificates.

3. The CA will check the submitted request against its policies. These policies could be about various things, like how much should be the minimum key length of the submitted certificates corresponding private key, or which particular encryption algorithm should be used. Furthermore, the CA or the RA may have some manual checks too wherein for example, the user may have to manually present himself at a point of contact with the CA or RA. If all of this succeeds, the CA can approve the certificate for issue.

4. Following successful approval for the certificate, the CA can issue the certificate, post it to a repository if the need be (depending on its own policies) and issue an identity certificate to the user. Now the user can successfully start using this certificate.

In the case of OpenPGP, a user can upload his or her public key to a key server. A public key received by the server is then either added to the server's own database or merged with an already existing key.

3.2.0.2 Distribution

In this case, the certificate is distributed to the user by the CA. It's considered a separate process than the Enrolment process, because some management functions of the CA might come into action. The CA can for example set its own policies that affect the use of the certificate. Furthermore, the certificate can also be distributed through a repository for other users to find the corresponding certificate for a user and use it to encrypt emails that they wish to send to that user. This could be a database or a directory service (using for example the Lightweight Directory Access Protocol or LDAP). The same repository also stores the Certificate Revocation Lists.

When it comes to OpenPGP, the certificates can be distributed using key-servers. This server holds OpenPGP keys that have been signed by the server itself. Other users can then request keys from the key-server. The server consults its own database and returns the requested public key if it is found, to the requesting user. The users may need to inform their own OpenPGP installations to trust such a key server as a trusted introducer. Hence, all keys signed by the server will be trusted by the user's OpenPGP installation.

3.2.0.3 Validation

When a certificate is used, the validation process for that certificate begins. In this process, the CA checks the status of the certificate and then checks that the certificate is not in the certificate revocation list (CRL). Some applications can also check the status of certificate before they use the certificate. It is however totally dependent on how the applications function and how they are configured.

Validation of a certificate differs highly in OpenPGP environments. As discussed before, validation is carried out using the web of trust model of OpenPGP along with trusted introducers.

Reason Code	Meaning
0	Unspecified Reason
1	Private Key Compromise
2	CA Compromise
3	Certificate User's affiliation has changed
4	Certificate or the private key have been superseded by a new certificate or private key
5	The issuing CA is no longer in operation
6	The certificate has been placed on hold
9	The CA has withdrawn the certificate user's privileges to use the certificate or the private key
10	Authority Information Access or AIA compromise

Table 3.1: The Reason Codes for Certificate Revocation

3.2.0.4 Revocation

A certificate that is issued by a CA has a time period for which it is valid. That is to say it has a start of validity date and end of validity or expiration date. At times, for various reasons, a certificate may need to be revoked prior to reaching its expiration date. In this case, the CA can be informed and the CA will add the certificate to its Certificate Revocation List or CRL. As mentioned earlier, various reasons exist for a certificate to be revoked. For example, the corresponding private key for a certificate was compromised or stolen by a malicious user. In this case the certificate owner can ask the CA to revoke his or her certificate. The CA has to provide for such functionality. Furthermore, while revoking certificates, the CA administrator needs to provide a "reason code" for revoking the certificate. The reason codes are presented in table 3.1.

PKI-enabled applications are configured to check CAs for their current certificate revocation lists. They mostly verify before using a certificate if it has been added to a revocation list or not.

The similar logic of revocation applies to OpenPGP certificates. However, the methodology is totally different owing to the web of trust model. As discussed in the Background chapter, any user who had signed a public certificate and had thus trusted the certificate, can revoke his or her signature from that certificate. This revoked signature holds as much weight as a revoked certificate does in a PKIX environment. If an OpenPGP certificate has to be revoked, only the owner or a user designated by the owner of the certificate can revoke the certificate. Furthermore, a key server can be used to inform other users about a revoked OpenPGP certificate.

3.2.0.5 Renewal

When a certificate reaches its expiry date, there are various options that can be taken. For starters, the certificate can be allowed to get expired. On the other hand, if the user still wishes to use the certificate, he or she can ask the CA to renew the certificate for them. The CA will of course have to approve this renewal request typically by following the checks on the user information again and checking its own policies for renewal. It has to be noted that in case of certificate renewal, the user can chose if they want to generate wholly new public-private key pairs or use the ones from the previous certificate itself. Furthermore, the renewal process can be automated or might require the user to submit a renewal request with the CA explicitly.

OpenPGP certificates have validity periods too which indicate when the certificate will expire. These certificates can be renewed as well.

3.2.0.6 Destruction

When a certificate is no longer in use, it has to be destroyed completely. Primarily this is done by deleting the certificate, its backups, and the backups of its private key as well as the original copy of the private key. Certificate destruction helps ensure that in the future the certificate is not used maliciously.

3.2.0.7 Auditing

The CA carries out certificate auditing. This is totally dependent on each CA. Each CA has a different set of management tools or even policies as to what all it wishes to audit. In general auditing of certificates involves, monitoring certificate creation, certificate expiration and certificate revocation.

Similar auditing of key servers can be carried out in an OpenPGP environment. Although this auditing is not as comprehensive as that in the case of CAs in PKIX environments, it still can provide details about when a certificate was enrolled, updated, revoked or requested by a user.

3.3 A Modified Certificate Lifecycle

Based on the above analysis of a certificate lifecycle as it looks generally, a more modified lifecycle that is appropriate for an effective certificate management service was created. This lifecycle looked at a certificate from its starting point till its end point while keeping in mind the certificate management service that this thesis wishes to design. Since this was a certificate management service and not a Certifying authority service, many of

the tasks that happen in the lifecycle of certificate did not apply and thus did not have to be implemented by the service. For example, the service does not have to create its own revocation lists or carry out the entire functions of revoking certificates. However, it should have the functionality to send a revocation request to the competent authority and initiate the revocation process.

Keeping all this in mind, and a vision for the certificate management service, a modified lifecycle was created and analysed.

1. User Registration
2. Request
3. Approval
4. Issue

   The above four steps of the lifecycle are similar to the steps described under enrolment in the previous section. However, here enrolment is considered as the next logical step after these four steps. So a user needs to register with the CA, then request for a certificate which the CA can approve or reject. Upon approval of the CA, the certificate is issued. In case of OpenPGP, when a user requests a key server to sign his or her certificate, the key server has to approve the request and then issue the resultant key server signed certificate to the user.

5. Enrolment – Once the certificate is issued, it can be enrolled. In PKIX environments, the CA can post it to a repository, and send back to the user an identity certificate that he or she can now use. In case of OpenPGP environments, the certificate is enrolled in the key server.
6. Distribution – Here the certificate is distributed to the user. It can also be stored in a repository and later distributed to other users through that repository. This could be a repository attached to a CA (for PKIX), a key server (for OpenPGP) or the repository attached to a certificate management service.
7. Modification – Here the certificate can be modified. Many a times a user may need to change some of their information in a certificate. This could be for example a name change and so on. Hence modification of the certificate should be possible. In this step, actually, revocation is followed by generation of a new certificate. While revocation is easy in a PKIX environment, in OpenPGP environments this certainly would require a key server. In the absence of a key server, users who have signed the certificate (whose data needs to be changed), will need to be contacted individually and they will have to revoke their individual signatures from the certificate.
8. Renewal – Here the certificate is nearing expiration and needs to be renewed.

9. Revocation – Here the certificate is undergoing revocation or has been revoked already. For PKIX environments, this would require the revocation of the CAs signature by the CA from a certificate. In case of OpenPGP, this would entail revocation of an individual or more signatures from an OpenPGP certificate or the revocation of the whole certificate by the owner or a designated revoker of the certificate.

10. Expiration – Here the certificate has expired and thus cannot be used any longer to encrypt or sign emails.

Looking at this modified life-cycle, various aspects of an effective certificate management service can be laid out. Figure 3.1, this life-cycle can be seen (Please note that although it is shown as a circle, it is not directional). In the next chapter, this thesis will use these very steps of the life-cycle to analyse the requirements for an ideal certificate management service.

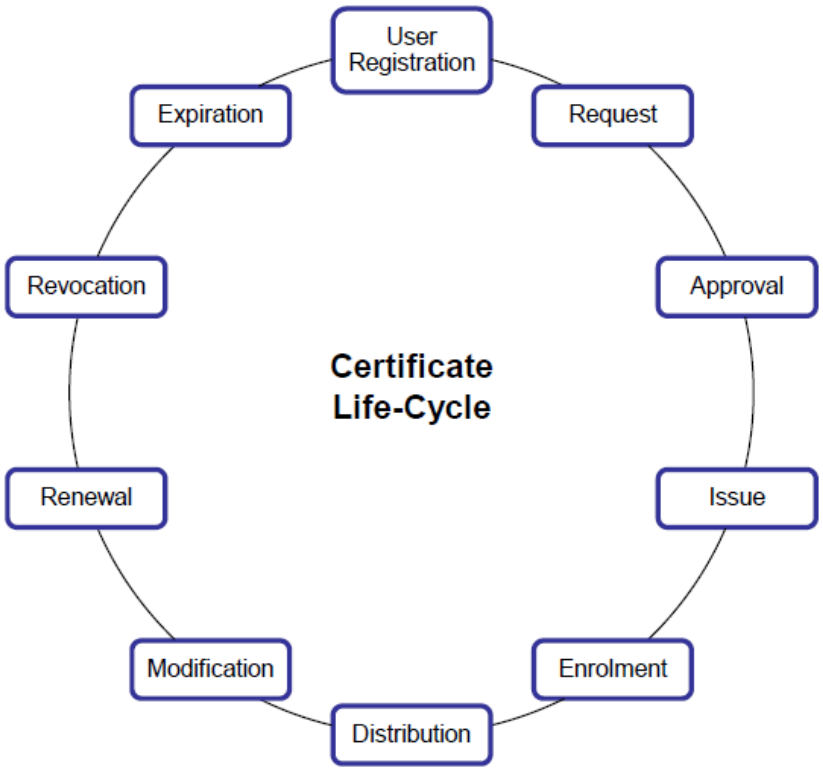

Figure 3.1: The modified certificate life-cycle.

3.4 The need for a Certificate Management System

It can now be clearly seen that the lifecycle of a certificate begins far before the certificate is issued and extends beyond the certificate being enrolled and distributed. Some of the steps in a certificate's lifecycle can also be complex to manage. Hence, an effective system is required which can manage a certificate throughout its lifecycle. To be truly effective, this system must take into account every step in a certificate's lifecycle. Furthermore, this system will also have some functionalities of an OpenPGP key server to accommodate the OpenPGP technology apart from supporting X.509 certificates.

This system must be able to register users who in turn can create certificate requests. The system should allow for a mechanism that allows for these certificate requests to be approved or declined. If and when a request is approved, the system should forward it to an appropriate certifying authority or registration authority to get a certificate issued for the approved certificate request.

Once a certificate is issued, this system should take care of certificate enrollment and distribution. The users using the system should be able to get their issued certificates easily through the certificate management system and its attached repository.

If a user requires, he or she should be able to modify his or her certificate data and the certificate management system should be able to provide the functionality for the same. The process of revoking certificates and issuing new ones with the modified data should be automated as much as possible.

The system should also provide users with an easy way to revoke their certificates. Furthermore, certificate renewal should also be feasible with an effective certificate management system. Lastly, in case a certificate expires, the certificate management system should present this information to the user.

Chapter 4

Related Work

There are a few systems that come to mind, when discussing certificate management services. These systems have existed for quite some time now. This chapter discusses three of the more prominent of these systems - Enterprise PKI with Windows Server, EJBCA and OpenCA.

All these systems have their own limitations, which will be discussed in their corresponding subsections. These limitations are a motivating force behind the need for designing and realizing a management service for digital certificates that is tailored to the needs of large organizations.

For starters as far as a certificate management service is concerned it is a service that can interact with these systems. While all three of these systems help in providing a PKI and a CA, there is still a need for a full fledged functional service that interacts with this. This service is not provided for by these systems.

4.1 Enterprise PKI with Windows Server

Enterprise PKI with Windows Server is not a standalone system by itself and is more of a service that can be configured on a Windows Server Operating System. Microsoft Windows Server can be used to set up a public key infrastructure. It can be done using the Active Directory Certificate Services or AD CS [27]. AD CS is an identity and access control security technology that provides for customizable services for creating and managing public key certificates that are used in software security systems that employ public key technologies [28]. By using the server manager in Windows Server, the following components of AD CS can be installed:

1. Certification Authorities - A root CA or subordinate CA can be set up to issue certificates and to manage the validity of the certificates.

2. CA Web enrolment - This allows users to connect to the CA through a web browser. Users can then request certificates and even retrieve certificate revocation lists.
3. Online Responder - This service accepts revocation status requests for specific certificates. It can evaluate the status of these certificates and can send back a signed response which contains the requested certificate status information.
4. Network Device Enrollment Service - This allows routers and other network devices to also be able to obtain certificates.
5. Certificate Enrolment Web Service - This allows users and computers to perform certificate enrolment and uses the HTTPS protocol.
6. Certificate Enrolment Policy Web Service - This enables users and computers to obtain certificate enrollment policy information.

Setting up the PKI with Windows Server is not that complicated. It can be used to setup the PKI in six logical steps as follows:

1. Build a standalone CA that will function as the root CA
2. Create an enterprise subordinate CA
3. Deploy certificate templates
4. Enable certificate auto-enrolment
5. Set certificate revocation policies
6. Configure and Verify private key archive and recovery

The setting up of the Enterprise PKI with Windows Server is not too complex although it requires a Windows Server operating system thus limiting the user-base. This is not a platform independent service. Furthermore another limitation is that AD CS supports variety of applications including S/MIME but does not support OpenPGP.

4.2 EJBCA

Enterprise Java Beans Certificate Authority, or EJBCA, is a free software public key infrastructure CA software package mantained and sponsored by PrimeKey Solutions AB [29].

EJBCA is implemented in Java EE. It has been designed to be platform independent. Multiple instances of EJBCA are run simaltaneously. They all share a database which contains the current certification authorities. Hence, each instance of the software can access any of the CAs.

EJBCA support various kinds of PKI architectures like single server, distributed RAs and external validation authority (A Validation Authority provides services that can be used to validate a certificate) [30].

EJBCA provides for an option to chose between SHA1 with RSA and SHA256 with RSA. It allows for different key sizes too. The supported key sizes are 1024, 2048 and 4096 bits. However, not all applications support the 4096 key size.

The prominent limitation of the EJBCA is that it does not support OpenPGP. The other limitation of EJBCA is that it is very complex to set up and configure. Although an EJBCA Virtual Machine, with the setup of EJBCA already done, is available from PrimeKey Solutions - it is a very basic implementation with many features of EJBCA still requiring further setup. Furthermore, problems have also been noticed with EJBCA when using 4096 bits key sizes.

4.3 OpenCA

OpenCA, officially called the OpenCA PKI research Labs, is a Public Key Infrastructure collaborative effort to develop a robust, well featured certification authority. It provides for implementing the most used protocols with full-strength cryptography. It is heavily based on various open source projects like OpenLDAP, OpenSSL, and Apache Projects [31].

OpenCA was built keeping one problem in mind - setting up of a public key infrastructure. Almost all of trust centre software is expensive. OpenCA's goal is to build the organizational infrastructure for a PKI. The databases in OpenCA are able to store all user related information from the point of view of a CA. This includes certificate signing requests, certificate revocation requests and certificate revocation Lists [32].

Almost all operations in OpenCA can be carried out using six pre-configured interfaces. Many more interfaces can be created from these interfaces. The cryptographic backend of OpenCA is OpenSSL.

OpenCA has been designed to work in a distributed infrastructure. It allows you to build a whole hierarchy of CAs and RAs with three levels or more. It can let you create an offline CA and an online RA. It can be easily used by both small and large organisations.

OpenCA is one of the best certificate authorities available to Linux users. Its basic features allow for various algorithms like DES, DES3 and IDEA. It easily supports large key sizes - 1024, 2048 and 4096 bits. The algorithm it uses for the digital signature is SHA1 with RSA encryption [30].

One of the prime limitation of OpenCA, just like the Microsoft Windows PKI is that it too does not support OpenPGP. Apart from this, unlike the Windows PKI, OpenCA

is difficult to setup and use. Though the developers of OpenCA have provided a well written user guide as well as an installation guide to do so, not every user can set it up. This limits its use to only expert users or organisations willing to part with funds to hire expert users to set it up for them. Furthermore, OpenCA cannot work as a standalone application. It requires interactions with its web interfaces to function.

Chapter 5

Requirement Analysis

To proceed further with the design of the certificate management service, a complete understanding of the requirements for such a service is required. Before the system could be properly designed, it was important to know what will be the various functions of the service. This analysis needed to encompass all aspects of the service. The requirement analysis built upon the analysis done in the Analysis chapter. This was important so as to make sure that the certificate lifecycle and the tasks related to it would be fully supported and integrated with the resultant system. The analysis of all stakeholders was chosen as the most suitable point to begin with. This was because, the stakeholder analysis could easily help determine every single kind of user that would eventually use the service. Then, from the stakeholder analysis, the various ways in which each stakeholder interacts with the system can be determined. This is synonymous with a use case analysis for each user. Following both the stakeholder analysis and the use case analysis, the functional and technical requirements of the system can be determined. Such careful analysis of the system ensures that all important functionality that should be present in such a system have been covered and not left out.

5.1 Stakeholder Analysis

The Stakeholder Analysis was the first of the two analysis carried out so as to understand the needs of the users that will interact with this system. The Stakeholder analysis looked at all the different kinds of users the system will interact with. Using the stakeholder analysis, the use case analysis can be carried out which can determine the various use cases for each user. In the Stakeholder Analysis, six types of users were determined – system administrators, end users, service operators, support staff, recovery operators and external users. Each user is discussed in detail below:

System Administrators – This is an administrative role similar to that of a system

administrator in other systems. They take care of the infrastructure in which the service runs. Their tasks are related to the environment in which the service runs and functions and hence they are interacting with the service completely "externally".

Service Operators – This is also an administrative role. However, the service operators handle the service directly and are interacting with the entire service "internally". For example, they can manage the users interacting with the service or create security policies which other users have to follow. Furthermore, they can monitor or interact with all the tasks that the service can do. Finally, they have to oversee that the service runs smoothly.

Support Staff - Just like a support staff at a call centre, they help end users with various kinds of support activities. For example, an end user may not be able to get a certificate issued by the service after repeated attempts. In this case, the end user can take the help of the support staff in fixing the issue. In brief, the support staff takes care of any questions and troubleshooting help the end users may require.

Recovery Operators – These users are designated persons whose job is to recover certificate keys. They have to take care of local regulations in this regard as well as the standards and directives of the organisation itself. For example, In Germany these keys are protected by Data Protection Laws, and this has to be taken into account by the recovery operators.

End Users – These are the main users of the service. They are the end entities that require certificates to communicate securely and for whom the service is being set up per se.

External Users - These users are not necessarily part of the organisation that is deploying the certificate management service. They are the users external to the organisation and with whom members of the organisation wish to and/or are already making secure communication using the certificates created using the service. They could be customers of the organization, research partners, etc.

5.2 Use Case Analysis

Based on the above Stakeholder analysis, use cases were analysed keeping each user in mind. Under the use-case analysis, each user's interactions with the system were noted down. This use case analysis is important as it will help lay the foundation for further analysis of work-flows for each user. The use case diagrams that were created as a result of the use case analysis are presented in Figures 5.1, 5.2 and 5.3. Figure 5.1 shows the use case diagram for the Service Operators, Recovery Operators and External Users. Figure 5.2 shows the use case diagram for the Support Staff. Lastly, the use case diagram for the End User is shown in Figure 5.3.

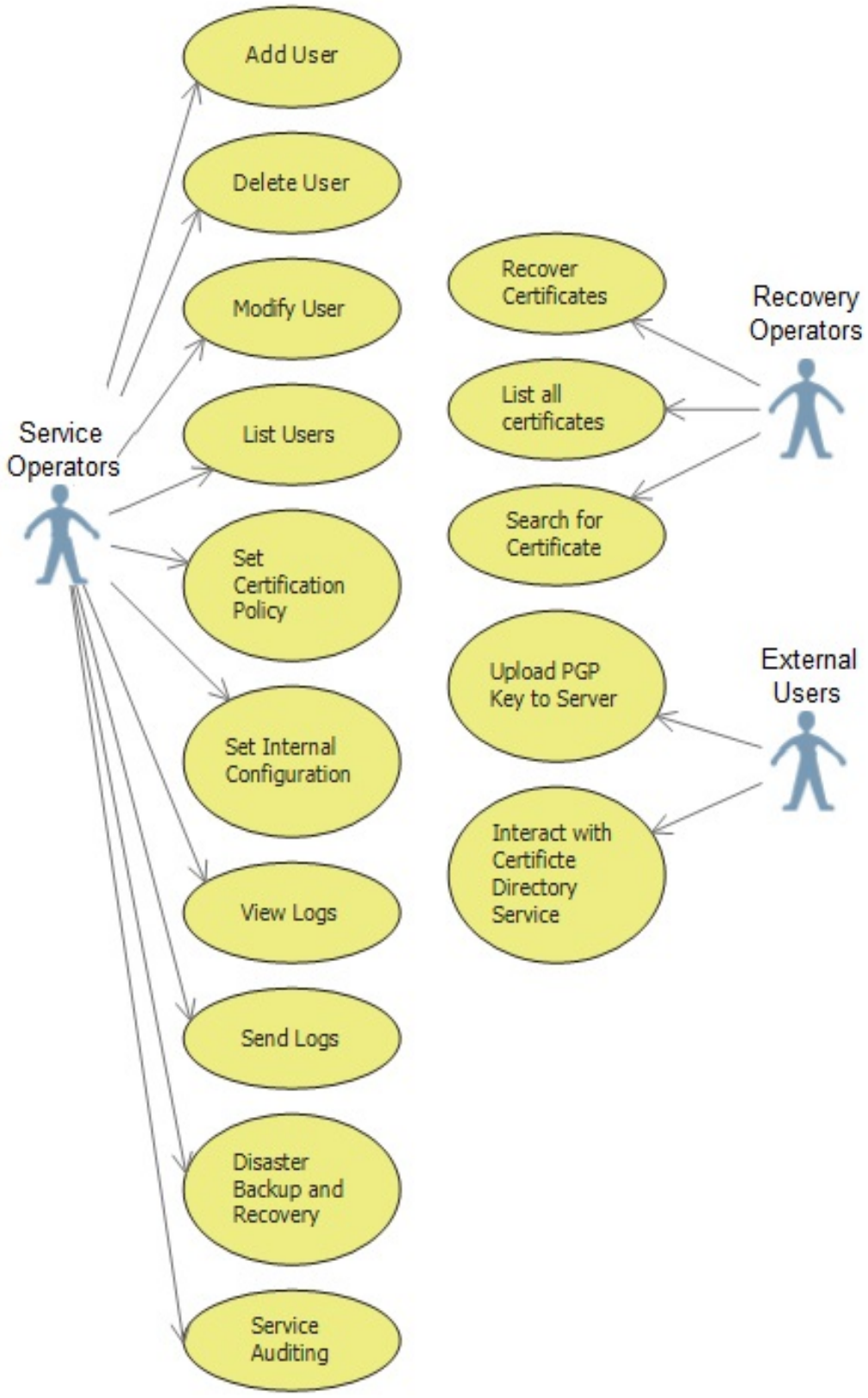

Figure 5.1: Use case diagram for the Service Operators, Recovery Operators and External Users.

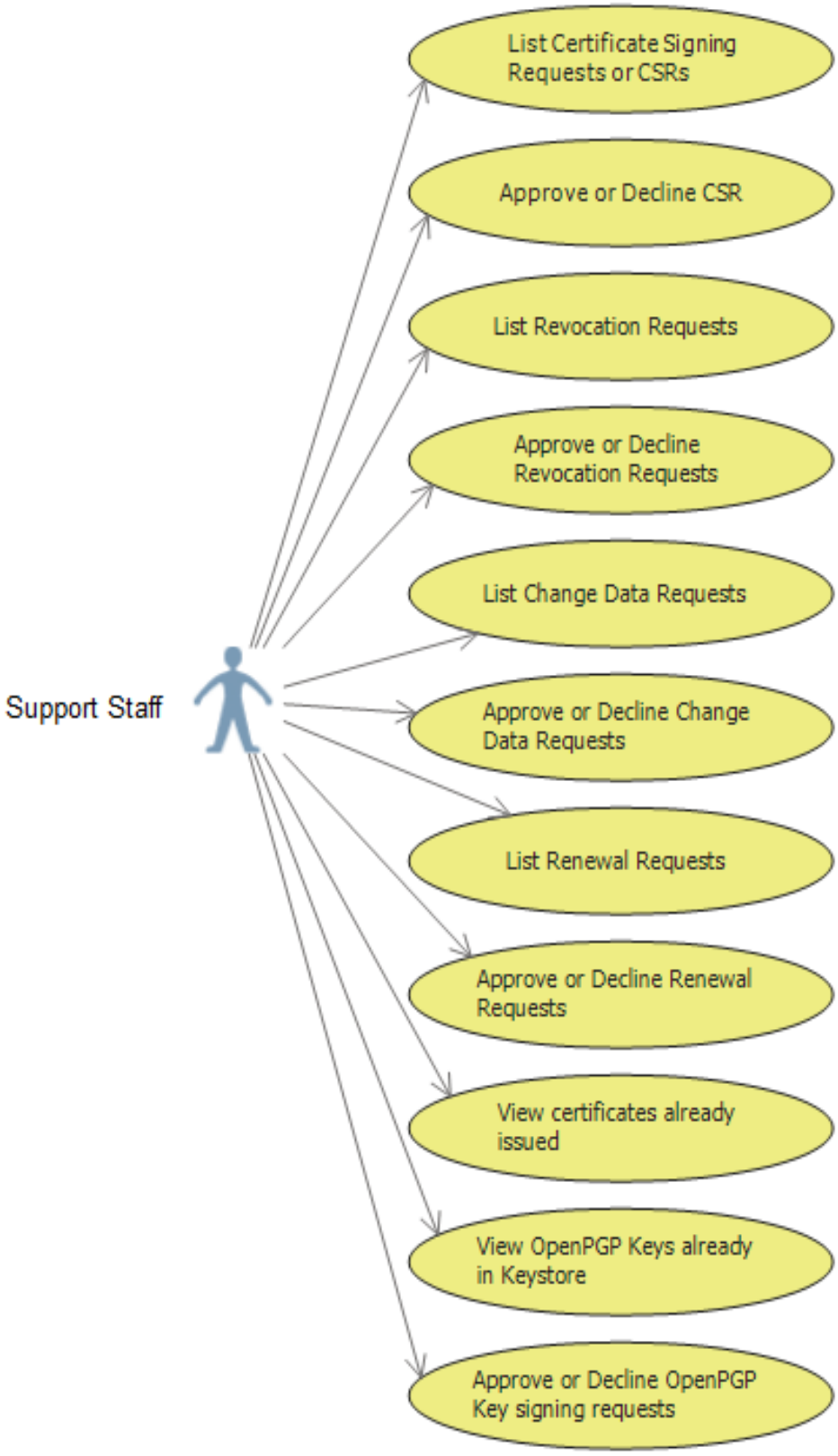

Figure 5.2: Use case diagram for the Support Staff.

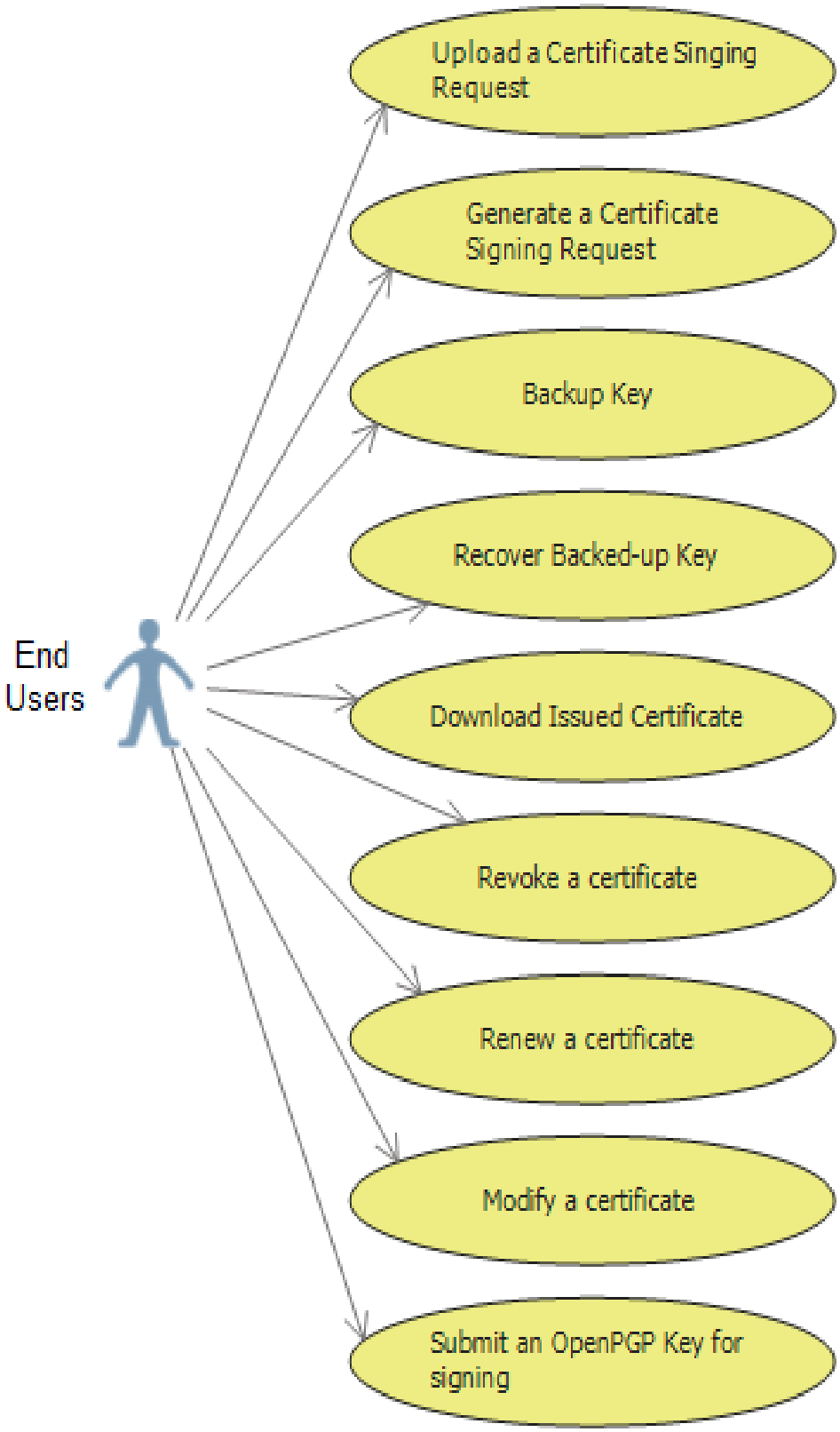

Figure 5.3: Use case diagram for End User.

Next, the use case analysis was further improved upon by a more holistic analysis. The results of this analysis, were required to determine the functional requirements of the service. There are multiple activity diagrams that will be presented in the next few pages. Though, divided for each user, some users have multiple activity diagrams showing different activities. This was done owing to space constraints.

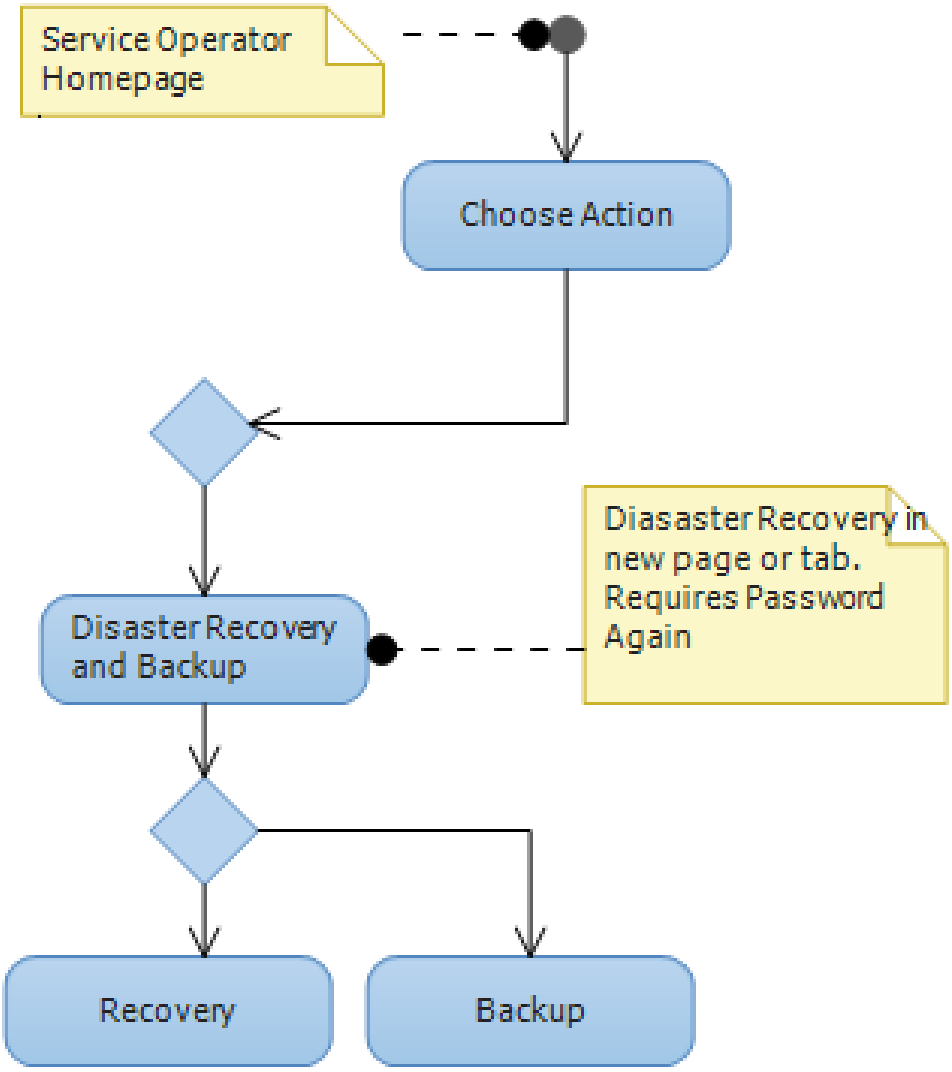

Figure 5.4: Activity Diagram for Service Operators showing their Disaster Recovery and Backup related work-flows.

In Figure 5.4, work-flows for Disaster Recovery and Backup have been shown. These work-flows are not highly detailed. The service operator can take a backup of the entire system. This includes generated certificates, certificate signing requests, log data, etc.

In case of a system failure, the service operator can recover the system using the

backups that have already been made. Added security can be provided by asking for a confirmation and asking for the service operator's password, whenever the recover function is to be used.

In Figure 5.5, basic user access management work-flows can be seen for service operators. As discussed earlier, service operators are the administrators dealing with the system internally. Hence, they take care of all user access in the system. In the basic sense, a service operator should be able to list all users that exist for the system. This can be seen by the **List Users** action in the work-flow.

Furthermore, the service operator must be able to create new users, delete existing users as well as modify user data for users present in the system. When the service operator shall list all users, he or she will have the option next to each user to delete them or modify their data.

User Roles are important for proper User Access Management. This shall help divide the users properly. The basic roles that shall exist in the system depends on the Stakeholder analysis done in the previous section. The roles are as follows:

1. Service Operators
2. Support Staff, and
3. End Users

Since the external users are to access the system without requiring a log-in, a role for them is not needed. The same applies for the System Administrators as they only interact with the system externally.

In Figure 5.6, work-flows for Reporting and Logging can be seen. The Service Operator has the task of reporting to the CA with audit logs of the system. In most cases, a CA may require various kinds of log data like logs of certification requests or revocation requests. It might be the CA's policy to do so or it might even be legal regulations that the CA is bound to follow. Thus, it is needed that the Service Operator should have an easy to use interface to easily create log reports, that he or she can submit to the CA. The Service Operator must be able to select what data out of the logs he or she wants to send to the CA. To further ease the work of service operators, it should be possible to select data by using various search options. Search options could include selecting what kind of log data needs to be selected, for which time period it needs to be selected and so on.

Apart from reporting purposes, logging is surely a great way for the service operators themselves to easily manage the system and audit it when needed. They can monitor system misuse and can also correct system faults if they have the logs readily available. The log data itself needs to be stored safely, as a CA can have a policy on data retention too.

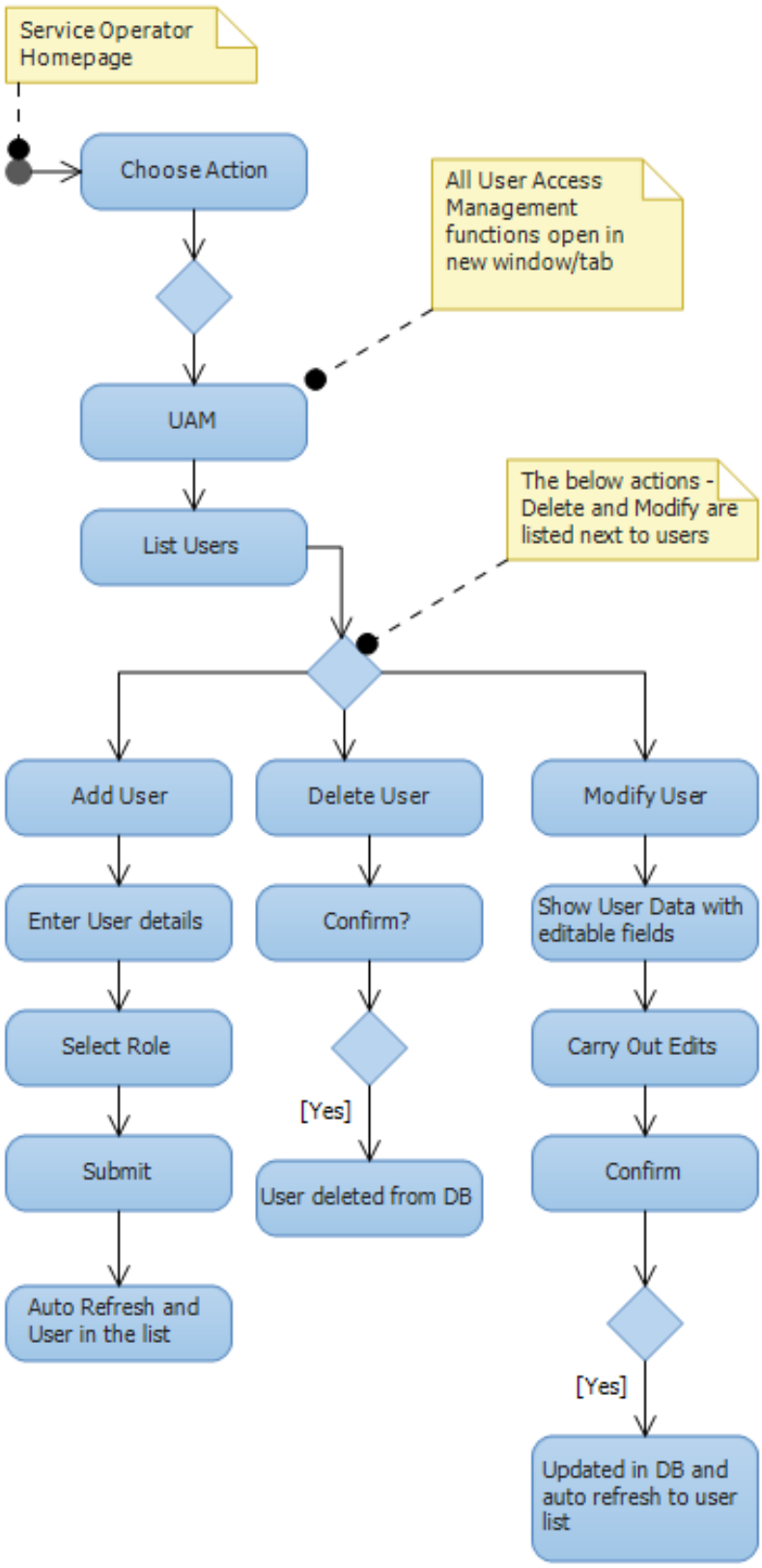

Figure 5.5: Activity Diagram for Service Operators showing their User Access Management (UAM) related work-flows.

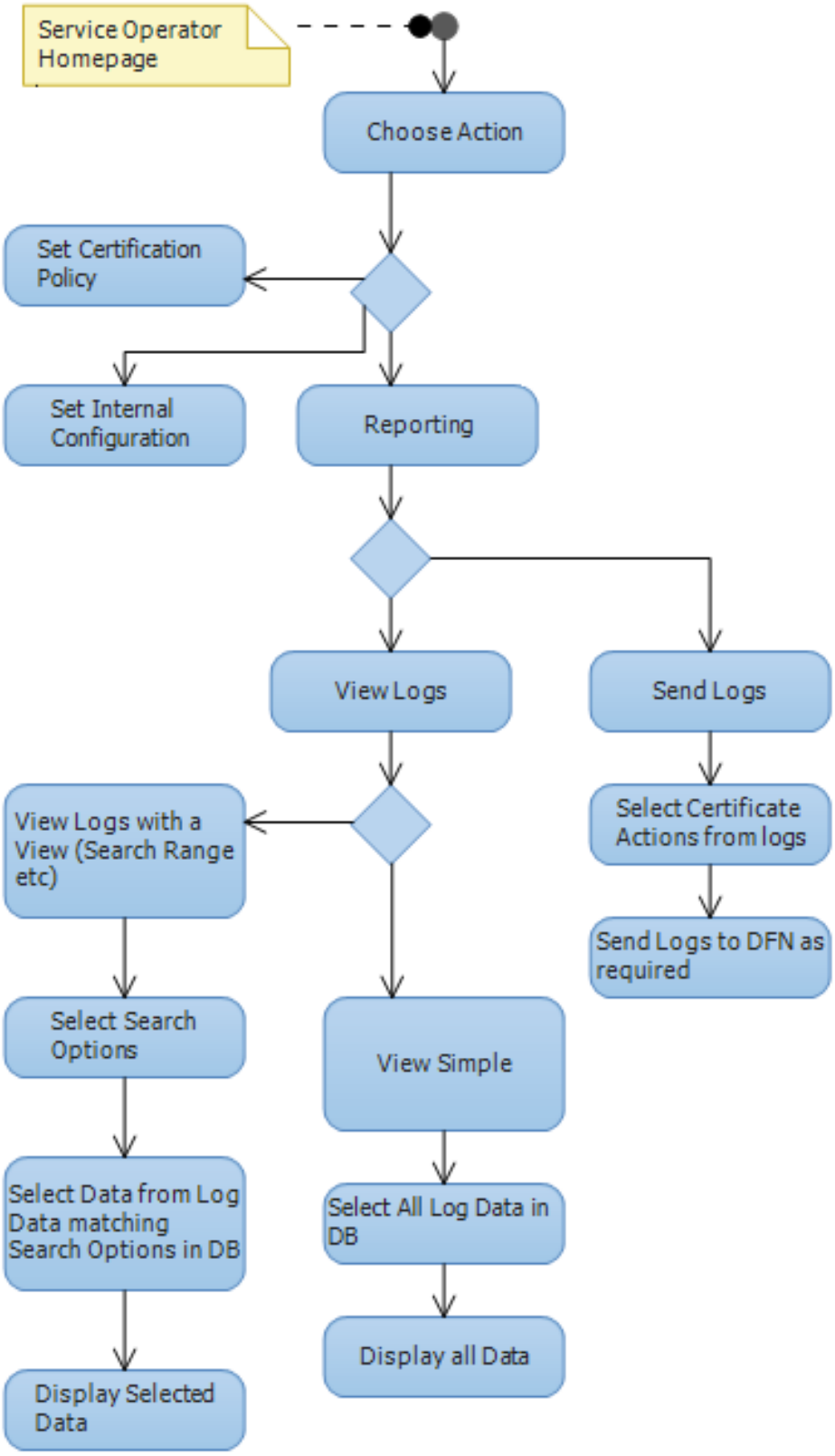

Figure 5.6: Activity Diagram for Service Operators showing their Reporting and Logging related work-flows as well as some other actions that they can take.

The service operators can also set the system wide Certification Policy. This could have various aspects to it like the minimum key length, or which algorithm to be used, etc. Service Operators will also be able to internally configure the system.

In Figure 5.7, work-flows for a Support Staff member listing and viewing certificate signing requests and revocation requests is shown. The information for the same is available in the database, and is then made available to the Support Staff members.

When it comes to the Certificate Singing Requests, the support staff still has a role to play. Once, the end user has sent in a Certificate Signing Request or CSR, physical checks may be required. For example, its oft the case, that the applying user will have to personally visit a contact point, where he or she has to present some kind of identity proof to prove that they are actually the ones applying for the certificate. This is where the support staff plays a role. A user will visit the support staff after sending a CSR to the system. There, the user will present some form of valid ID to the support staff.

The support staff can then approve or reject the user's original request. This request will then be sent automatically to the CA. The CA can have its own checks and policies to take care of. Once done, the CA can issue the certificate (which will be directly sent to the user for download) or the CA can reject the request.

When it comes to the revocation requests, the support staff plays a role here too. Upon receiving a revocation request, the support staff can cross check the details and approve the requests.

A provision for auto approval of revocation requests shall exist too. Here, if a revocation request is not approved for say 48 hours, it is automatically approved. For logging and reporting purposes, as well as for the purposes of the Certificate Revocation Lists maintained by a CA, a proper reasoning for revocation of the certificate must be given. An end user may not be technically sound enough to provide these details. Hence, the support staff will have the option to vet the request, read the user provided description and then select an appropriate reason code for the revocation from a list.

In Figure 5.8, we can see the work-flow related to changing a certificate's data. For various reasons, a user may apply for changes to his or her certificates data. This change of data should and will have to be approved by the Support Staff. This change in data could once again require a Physical Identity check.

The support staff shall be able to approve a change data request on his or her discretion. Once approved, the system will revoke the previous certificate with the old data. Simultaneously it will also issue the new certificate with the new data and update the database for the same. The new certificate can then be sent to the user.

The figure also shows the work-flow for approval of renewal requests and OpenPGP key signing requests.

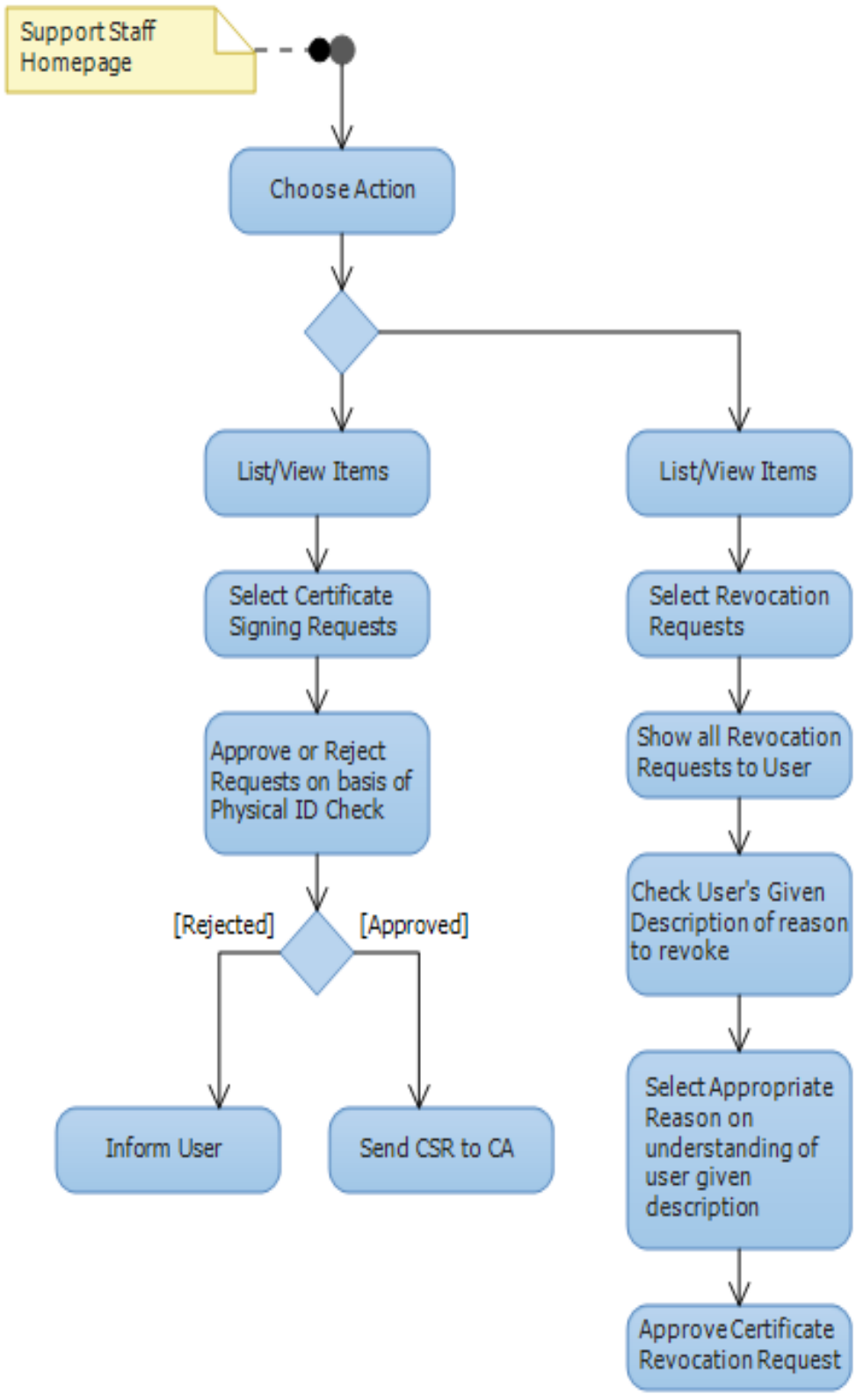

Figure 5.7: Activity Diagram for Support Staff showing work-flows related to Approving Certificate Signing Requests and Approving Revocation Requests.

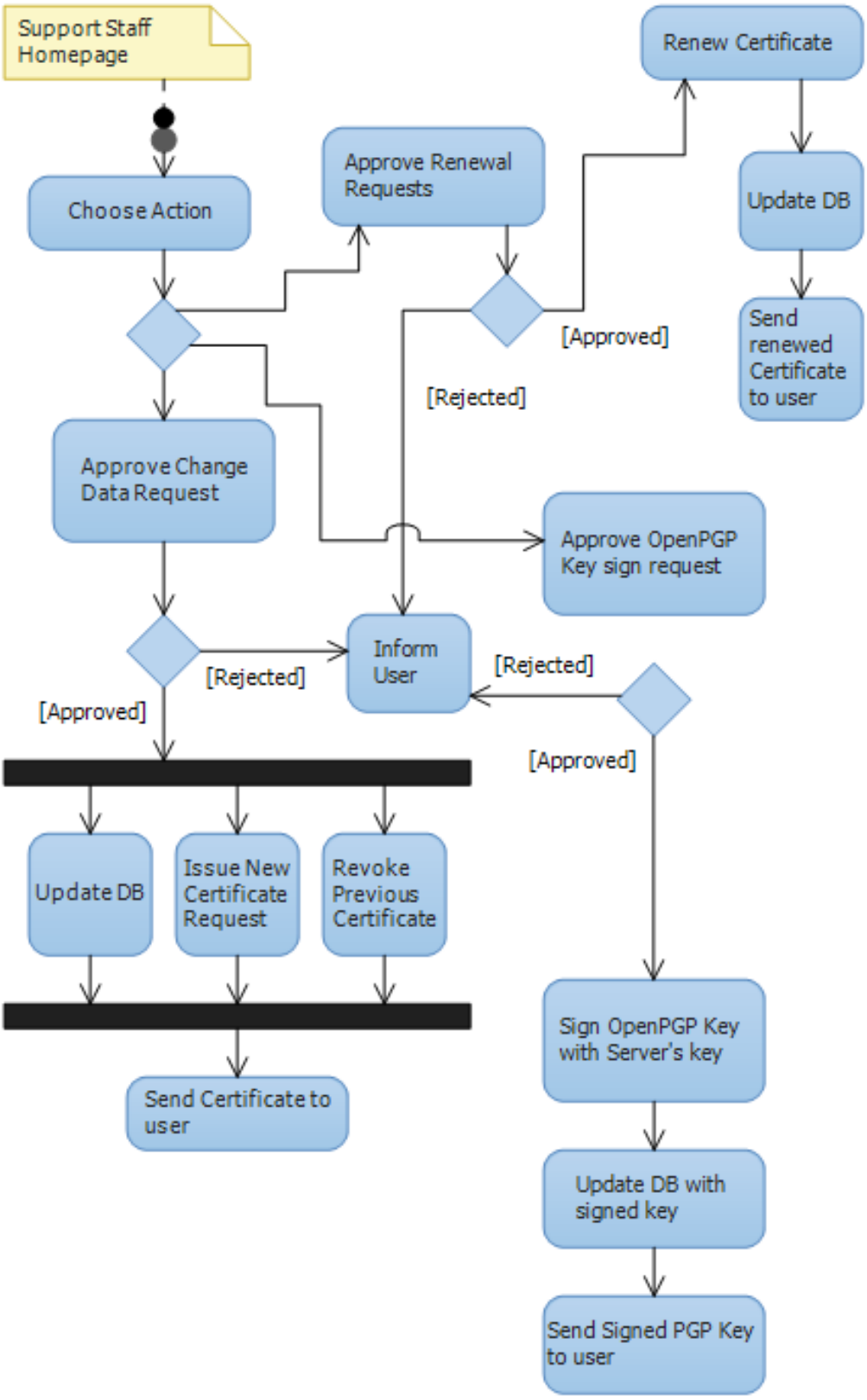

Figure 5.8: Activity Diagram for Support Staff updating a user's information in a certificate that was already issued.

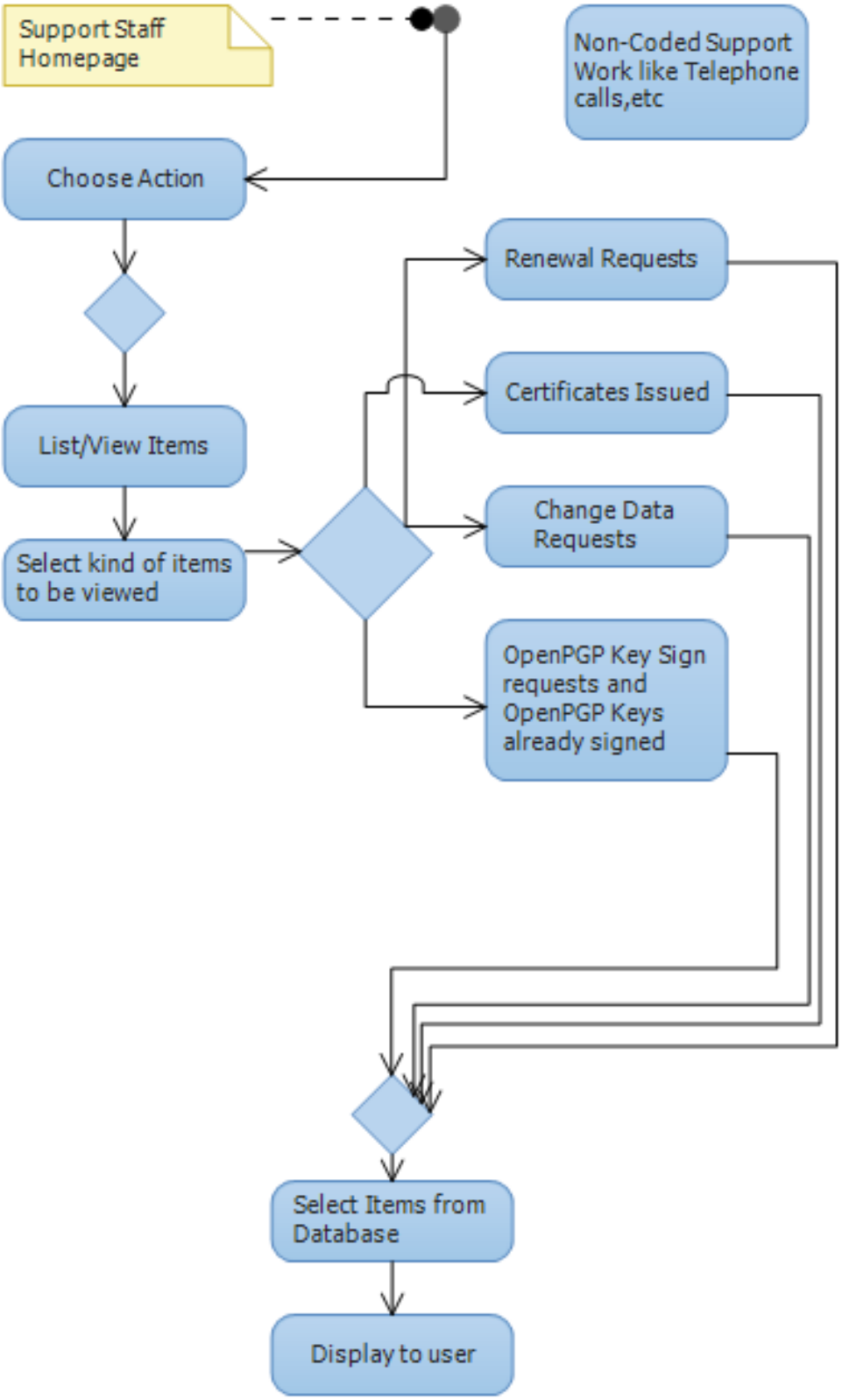

Figure 5.9: Activity Diagram for Support Staff's various work-flows where they can query the database for various kinds of information.

In Figure 5.9, a bit more simpler work-flows for the Support Staff can be seen. In short, these work-flows discuss the various kinds of information the support staff can see about the system and about user activity within the system. This information is the additional information that can be seen by the Support Staff, apart from Certificate Signing Requests and Revocation Requests. The additional information is:

1. Renewal Requests - All requests made by users to renew expiring certificates.
2. Certificates Issued - All certificates that were successfully issued by the CA. Before the certificates are transmitted back to the users, they need to be stored in the database of the service as well.
3. Change Data Requests - All requests made by end users to change their validation data.
4. OpenPGP keys present in key server and keys uploaded to the server - This shall have information about all OpenPGP keys that have already been uploaded and successfully signed by the key server's key. It shall also includes all requests made to upload OpenPGP keys by users to the server.

The above described functionality can help the support staff provide better support to other users. Do note that the support staff can also be contacted by E-Mail, Phone or Physical visit. The additional means of communication are important to improve the reachability (accessibility) to the Support Staff, else their role in the system will curtail.. The Support Staff can also be contacted by external users, for example if an external user feels that a certificate is being misused.

In Figure 5.10, the recovery staff's work-flows are shown. The recovery staff should have the ability to search for a user and then select the certificate that the Recovery Staff may wish to recover.

Since this matter concerns the local regulations, organizational policies, etc. these work-flows need to be closely monitored and have checks and balances in place. Many a times a Data Protection Officer will be informed as well. Without their sanction, these tasks may be nearly impossible to crack.

If the work flow is to be looked at, it is very straightforward. The recovery operators can list all certificates that were issued through the service. Since the database stores all information regarding CSRs, Issued certificates, etc. this data is always available. Upon selecting a certificate, the recovery operator can view the certificate's public details. There should be an appropriate function or button next to each certificate that kick starts its recovery process. Once this process is started, the Data Protection Office or DPO is contacted about recovery of the said certificate's private key. Alternatively, the recovery operator can also search for a certificate, instead of going through the tedious task of finding one in a huge list.

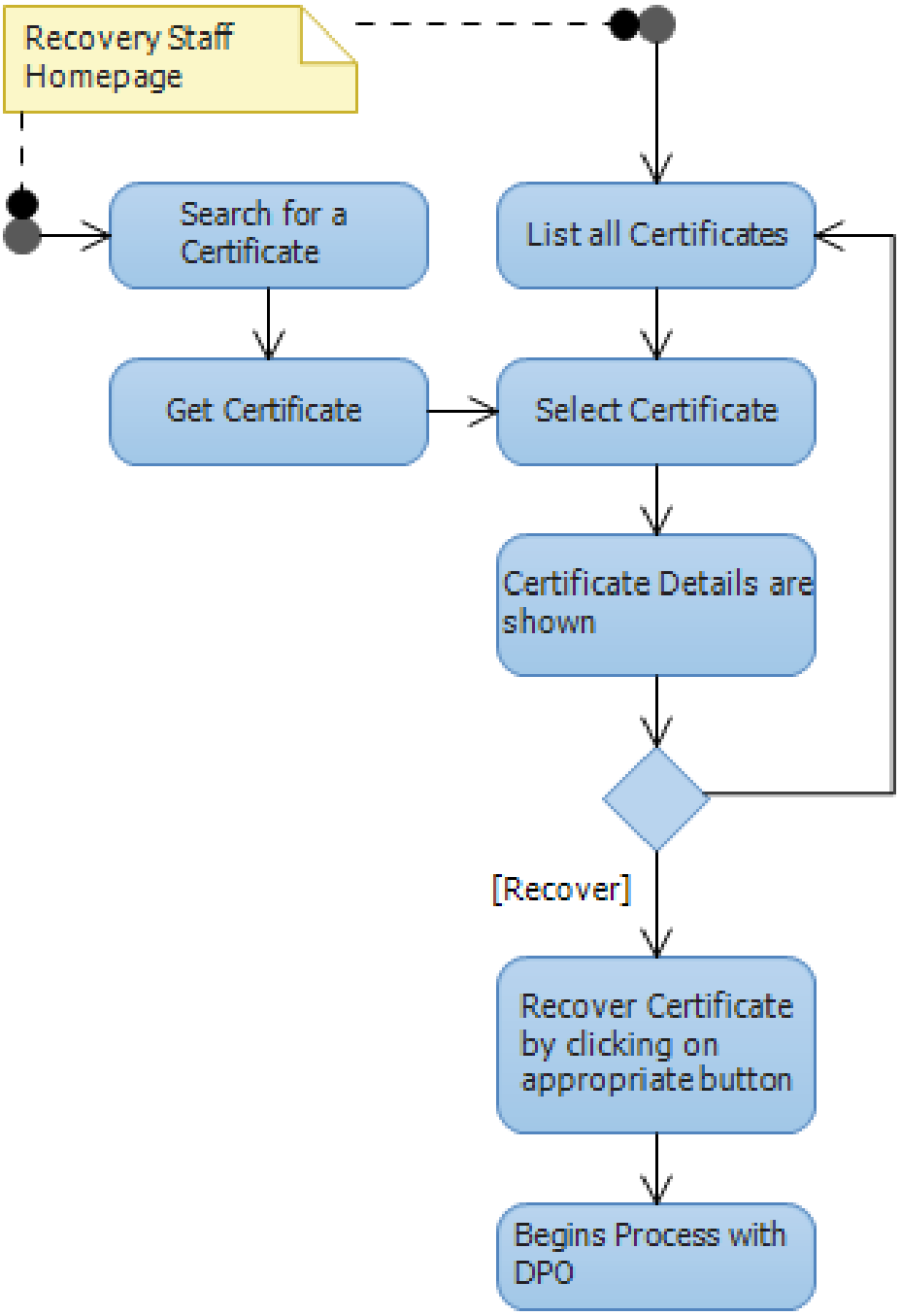

Figure 5.10: Activity Diagram for Recovery Operators.

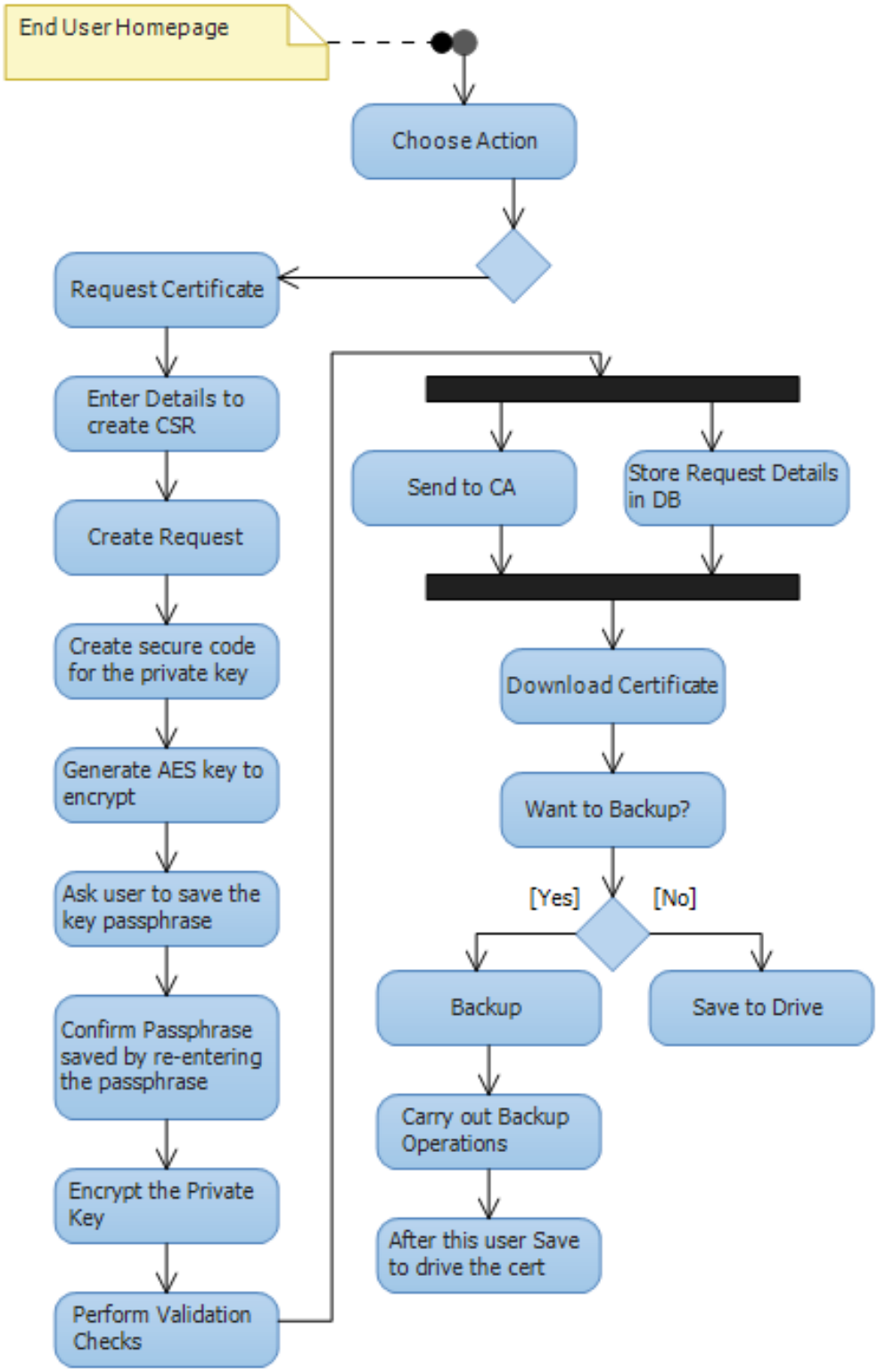

Figure 5.11: Activity Diagram for an End User showing work-flows related to certificate requests where a CSR is generated within the browser itself.

In Figure 5.11, the detailed activity diagram shows the entire work-flow of generating a Certificate Signing Request in a browser.

The end user is able to create a CSR in the browser, by providing all the needed details. These details include the needed fields for a X.509 certificate. Once the user clicks the generate CSR button, a CSR is generated along with the corresponding private key and public key. The user should have the option to backup the private key to the server. Before being transmitted to the server, the private key is encrypted to prevent it from being compromised.

In a new window, a secure code for the private key will be generated. This will be generated by the service itself. The user shall be prompted to save this secure code, and then in the next window asked to provide it again. This way it can be ensured that the user has saved the code somewhere locally. Then this code will be used to symmetrically encrypt the private key. Once encrypted, the encrypted blob can be sent to the server for storage.

The service will need to carry out validation checks on the data in the CSR. This could include checks on the CSR following the security policies as laid down by the Service Operators. Apart from this, manual Identity checks may be also required as per the CA's policy. In case of this certificate management service, a support staff member may need to carry out a manual identity check to approve the CSR and then send it to the CA. The CSR request needs to be stored in the database of the service. The CSR should be ideally stored along with the public key. When the certificate is issued (if it is issued) by the CA, it should also be stored along with the corresponding CSR and public key. If the user backs up his or her private key to the server, then that should also be stored correctly with the corresponding CSR, Public Key and Certificate.

In Figure 5.12, the work-flow for submission of a revocation request can be seen. The figure, also shows, the work-flow for an End User to change his or her certificate data.

The work-flow for creating a revocation request is straightforward. While submitting the revocation request, the user has to supply related details. Primarily this would include a description of why he or she wishes to revoke their certificate. This request, upon confirmation, is then sent to the server, where the Support Staff needs to approve it before it is sent to the CA. Note that the revocation request shall auto approve in 48 hours, if no action is taken by the support staff. Once the revocation request is complete, the user can be informed. In future, as soon as the certificate is revoked, the user should be informed about that too.

If the user wishes to change his or her validation data, then in that case, all they have to do is enter the new data and submit it to the service. The Support Staff once again comes into play, as they may have to perform manual identity checks once again. Note, that the previous certificate will be revoked and a new one will be issued, as discussed under the appropriate activity diagram for the Support Staff.

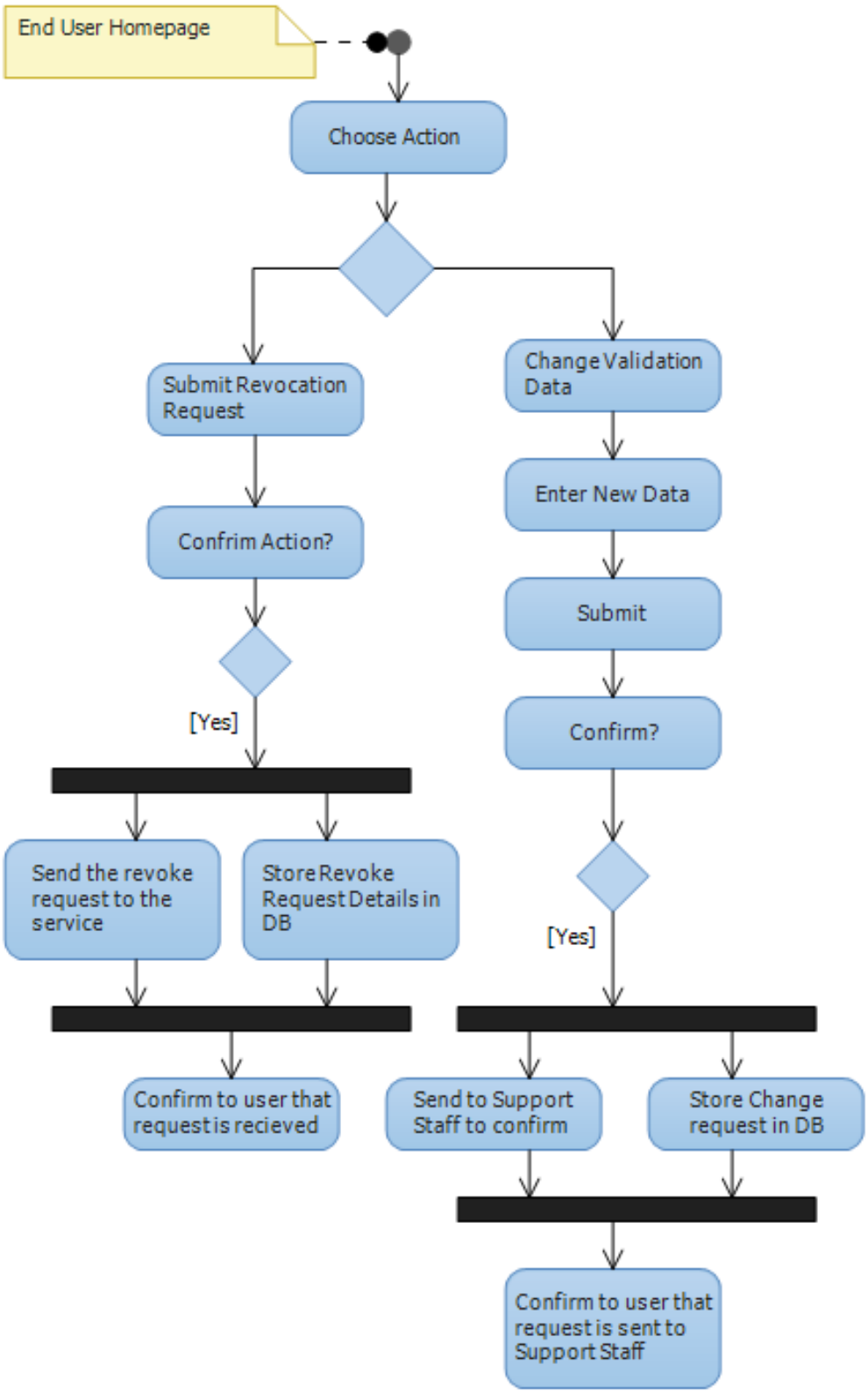

Figure 5.12: Activity Diagram For an End User showing work-flows related to submitting of requests for revocation or requesting for change in certificate data.

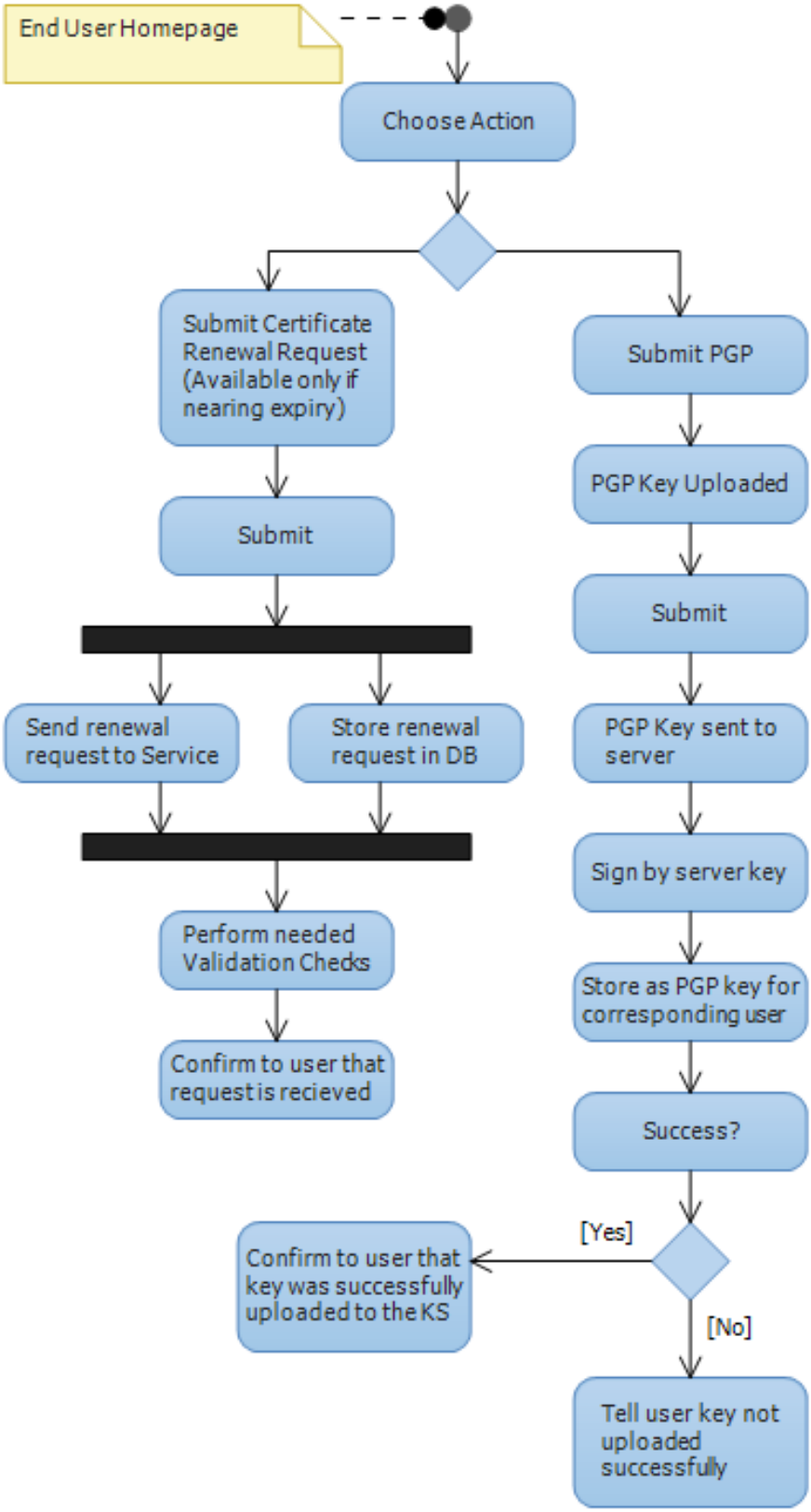

Figure 5.13: Activity Diagram For an End User showing work-flows related to submitting of certificate renewal requests and submitting of OpenPGP keys.

In Figure 5.13, the work-flow for submitting a certificate renewal request can be seen. The work-flow for submitting a OpenPGP key to a key server is also present.

When a certificate is nearing its expiry, the end user shall have the possibility to fire off a renewal request to the service. This renewal request is simple. The normal validation checks and verification of user data, as done when creating a certificate for the first time will need to occur again. These checks are needed to assure if the user still requires a certificate or not. For example, the user could have moved out of the organization and hence no longer should be issued a certificate. Once again, manual identity checks may also be required. Once the support staff approves, the renewal request, it can be sent to the CA.

It needs to be noted here, that a methodology of automatically warning users can be created when their certificate is nearing expiry. This could be an internal service-wide warning system or an external system that for example triggers an E-Mail message to a user informing them that their certificate is expiring shortly.

In case of submitting a OpenPGP key to the key server (KS) for storage, the work-flow is simple. The user uploads his or her OpenPGP key which needs to be signed by the key server's key. This shows that the key server trusts the user's key. Once signed, it needs to be stored on the key server, and mapped to the corresponding user.

In Figure 5.14, some more work-flows can be seen. The End User has the option to also upload an already generated CSR to the system. This CSR may be generated using the OpenSSL command line tool for example. Once the CSR is uploaded, it needs to be parsed and data in it needs to be verified before it can be sent to the CA for issuing a certificate. Here too, the validation checks come in to play as they did with a CSR being generated in the browser. The manual identity checks may also be required.

The End User, should also be able to back up his or her private key to the system. This option is provided, in case the user forgets to backup his private key while generating the CSR in the browser. This option also helps in the case where a user maybe generating a CSR locally and uploading it to the service. However, in this case some checks are needed to be performed - primarly if the private key being backed up actually corresponds to the public key information already available with the service.

Lastly, the End User, has the functionality of recovering a lost key from the system, provided an appropriate back up exists. This is a simple interaction with the server. The user should have his or her secure code safe somewhere, else once the backed up key is sent to the user, it will be unrecoverable as it will be an encrypted blob. The secure code generated during backing up the key will be needed to get the actual private key back.

These were all the activity diagrams for the stakeholders that are needed to be properly authenticated to access the certificate management service. The next stakeholder - the external user - is different in this regard.

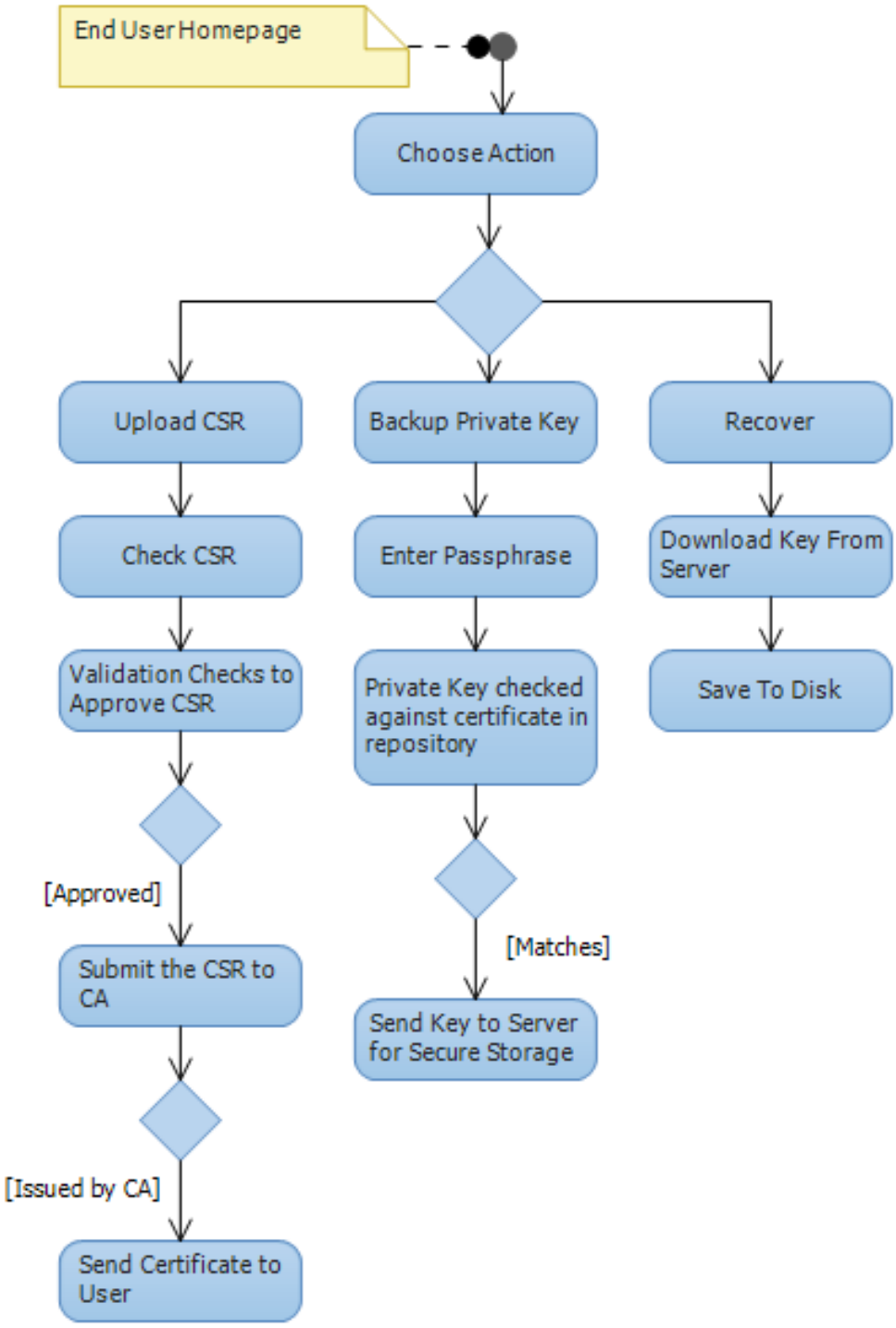

Figure 5.14: Activity Diagram for an End User showing work-flows related to Uploading of a CSR, Backing up of private keys and Recovery of backed-up keys.

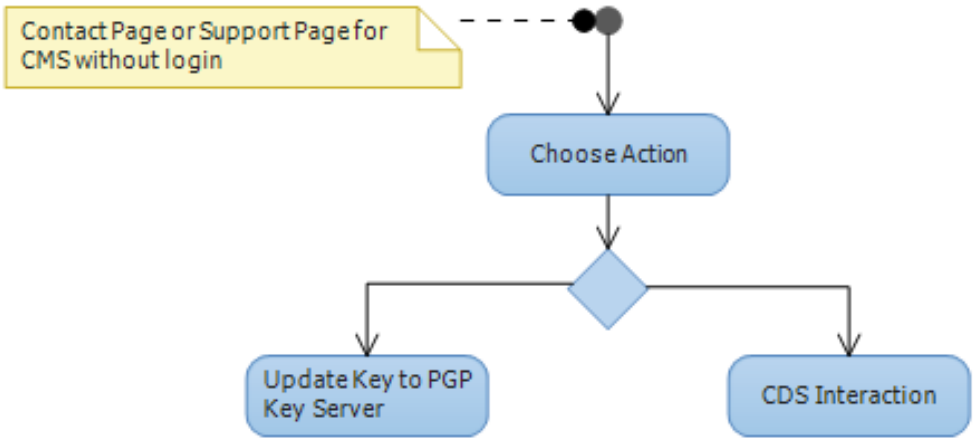

Figure 5.15: Activity Diagram for External Users.

In Figure 5.15, a simple activity diagram is presented. The external user does not require to be logged in to the system. The external user can update the OpenPGP Key server with his or her own key. The external user can also interact with the associated Certificate Directory Service of the system.

5.3 Functional Requirements

Based on the Use-Case analysis, the functional requirements could be determined. The following list shows these requirements. This covers the major functional requirements the system needs to have. It also lists some non-functional requirements of the system. (Note that FR stands for Functional Requirement and NFR stands for Non-Functional Requirement).

- FR1: Upload Certificate Signing Request - Once uploaded, to let the certificate be checked and validated and then sent to the CA for issue. Once issued, the certificate should be sent back to the user.
- FR2: To generate a CSR in the browser - In case the user does not have a CSR already generated, he or she can do the same in the browser and then send it to the CA through the service.
- FR3: Backup and Recover Key Data - Users have the ability to back up and later recover their key data if for some reason they would lose their own local copy. This should also allow to back up the key data when not done initially (For example, when the CSR was generated in the browser).
- FR4: Revoke certificates - Users should be able to send revocation requests to the

service which can then be vetted by the Support Staff before being transmitted to the CA.

- FR5: Renew Certificates - Users should be able to renew their certificates which are going to expire soon.
- FR6: Change Certificate Data - Users should be able to change their certificate validation data if ever the need arise.
- FR7: Submit OpenPGP Keys to Server - Users should be able to submit their OpenPGP keys to the OpenPGP Key servers.
- FR8: Request Certificate Destruction - Users must be able to request the complete destruction of all their certificate related data available with the certificate management service.
- FR9: Logging and Reporting - Service Operators shall be able to easily maintain and manage logs and reporting functions as required by the CA.
- FR10: User Access Management - Service Operators must be able to easily manage the users of the system.
- FR11: Back-up and Recovery of the System - This should be possible for the Service Operators to carry out.
- FR12: Set Internal Configuration - The Service Operators must be able to set up the internal configuration of the system.
- FR13: Set Certification Policy - Service Operators must be able to set up the system-wide certification policies.
- FR14: View and Verify requests in the system - Support Staff must be able to view all pending requests in the system.
- FR15: Recover Keys - Recovery Operators must be able to recover keys.
- FR16: Interact with Directory Services - External Users must be able to interact with the directory service.
- FR17: Report Issues - All users must be able to report issues to the Support Staff.
- FR18: Report Certificate Specific Issues - External users, not logged in to the system, must be able to report certificates they may think are being used maliciously.
- FR19: Update Key to OpenPGP Keyserver - External Users must be able to have this basic interaction with the key server.
- NFR1: Usability - The system needs to have a high emphasis on usable security.
- NFR2: Underlying Database - The system should be able to easily switch between different databases.

- NFR3: Cryptography - The system should use strong cryptographic algorithms and ideas to keep all information in the system especially the backed up private key data secure.

5.4 Technical Requirements

On the basis of all the above analysis, a technical requirement analysis was also needed to be carried out. Through this, the various technical requirements that are needed for the service to run smoothly could be determined. These are the basic requirements for the environment in which the certificate management service is to run.

The certificate management service requires a persistent storage for its database. It would also require an interface with an external CA, as the service itself does not carry out the functions of a CA. The service will also require an interface to be easily accessed externally. This would be best achieved by realising a REST (Representational State Transfer) API for the service. This API should take into account that external access to the service should be secure.

Chapter 6

System Design

This chapter discusses the system design of the certificate management service. On the basis of the functional requirements and technical requirements laid down in the previous chapter, an actual system can now be designed.

6.1 A Reliable Design

The certificate management service is to be designed so that it can effectively cover all requirements of such a service while allowing the functionality provided by the service to be accessible through external components.

6.1.1 System Components

The certificate management service will be best realised if it was divided into two main sub-components – a frontend and a backend. These are the primary internal components of the service. This separation simplifies development of the service and helps ease future maintenance work. HTTP calls can be easily made from the frontend (client-side) to the backend (server side). Furthermore, a uniform interface is required to allow for interactions between the client and server. This should be realised by using HTTP Verbs (like GET and POST), Uniform Resource Identifiers and HTTP Responses.

The frontend (or the CMS Frontend) will be the entry point for the certificate management service. This will be the part of the service that is facing the users. It is the client side of the certificate management service. It interfaces with the backend to provide functionalities to users. This component accepts user input and forwards it to the backend for processing. It also fetches processed data from the backend. The frontend takes care of presenting data that has been processed by the backend to the user.

The Backend (or the CMS Backend) is the server side part of the service that processes data and generates output after processing user input (received through the frontend). The output generated in the backend is sent to the frontend for presentation to the user. Hence, the backend does not have to worry about data presentation. Owing to this, the backend functionalities can be implemented independently of the frontend. As long as the interfacing between the frontend and the backend is done right, independent development of both components is possible.

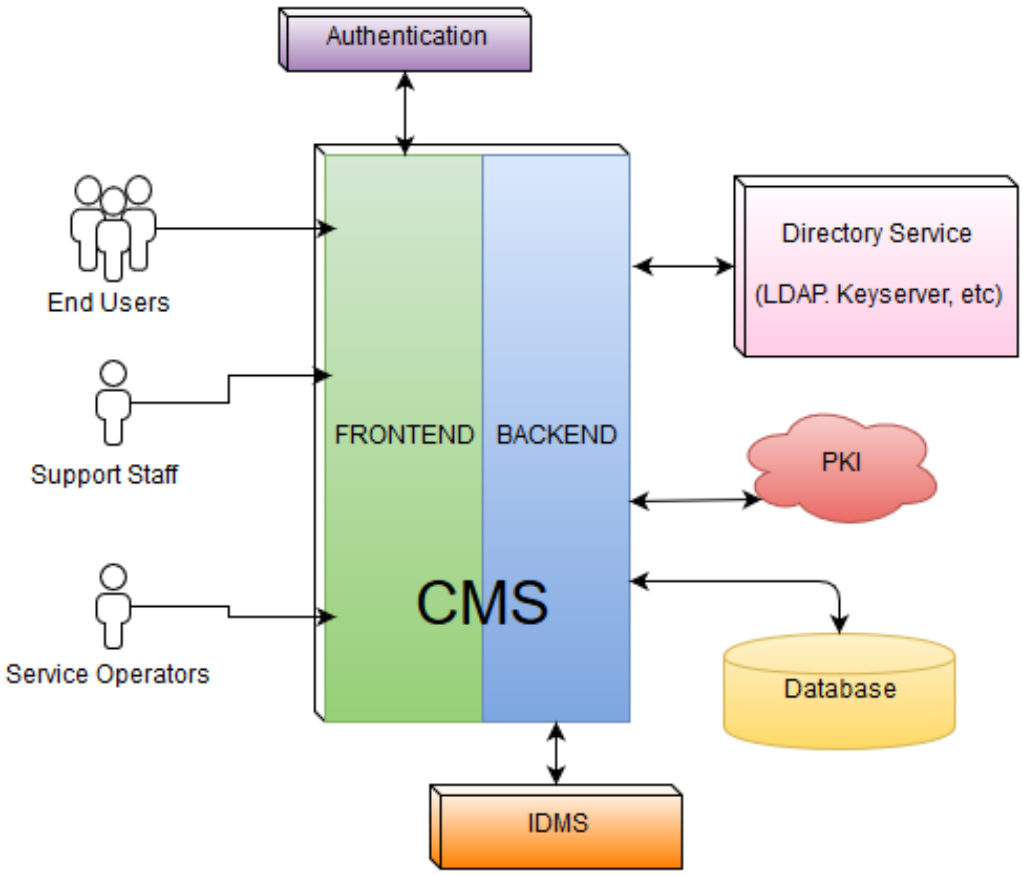

Figure 6.1: The component diagram for the certificate management service.

As can be seen in the figure 6.1, a simple design for a certificate management service is now created. The service has one primary internal component. This is the Certificate Management System itself. It is the central component in the service. It includes two sub-modules - the CMS Frontend and the CMS Backend.

Various other external components are also seen in the diagram. A brief description of all the external components is as follows:

1. Database Storage – This is the database in which the service stores all service-generated and related data. Requests made by users, need to be stored for further processing and also to help in audits of the service. Hence the database storage

is an important component of the service. Interactions with this component are controlled by the backend.

2. PKI Infrastructure – This includes the RAs and the CAs. The certificate management service needs to forward requests made by users to an appropriate RA(s) or CA(s). For example, a certificate signing request, whence created by a user inside the certificate management service, needs to be sent for approval to a CA. Interactions with this component are also controlled by the backend.

3. Authentication component – This module takes care of authenticating users into the service. This component would ideally be used, if for example members of an organization need to use their organization specific authentication credentials to log in to the service. Hence, the certificate management service should be able to authenticate users using these credentials. Ideally, the authentication component interacts with the frontend of the service to authenticate users.

4. IDMS (or) Identity Management System - This is also a organization specific component, that is to be used by Support Staff to verify identities of members of the organization that wishes to deploy the certificate management service. This component also interacts with the frontend of the service.

5. Directory Service - This is also an external component for the service. This service interacts with the certificate management service, so as to be able to get the certificates that have been issued by the certificate management service and make them readily available to users through its directory services. This could be a simple key-server that stores OpenPGP keys or a LDAP directory. The Directory Service relies on the database component.

6.2 CMS

The CMS is the Certificate Management Service. It is the central component in our service. As already discussed in the technical requirements of the previous chapter, the service should be realised as a RESTful web application. In line with this architecture, this component is divided into two sub-modules - the CMS Frontend and the CMS Backend.

6.2.1 CMS Frontend

The front-end of the service provides an entry point for a user (or another application) that wishes to interact with the service through a graphical user interface required for the service (or through a RESTful API). It also inculcates various security mechanisms to make the interaction with the service secure. This component is open to input which can

then be further processed by interacting with the backend. Furthermore, this module interacts with an external component, namely the Authentication component, which takes care of User Access Management.

All the proposed functional requirements need to be inculcated into the frontend. An interface to the users is provided by the frontend, so that they can carry out all of the functions discussed in the previous chapter.

The frontend also needs to provide for an entry point for all kinds of users to carry out their respective functions. The design of the frontend has to thus separate functionalities by using an authentication mechanism for users. Therefore, a user with a certain role will only be able to see the functions available to that particular role.

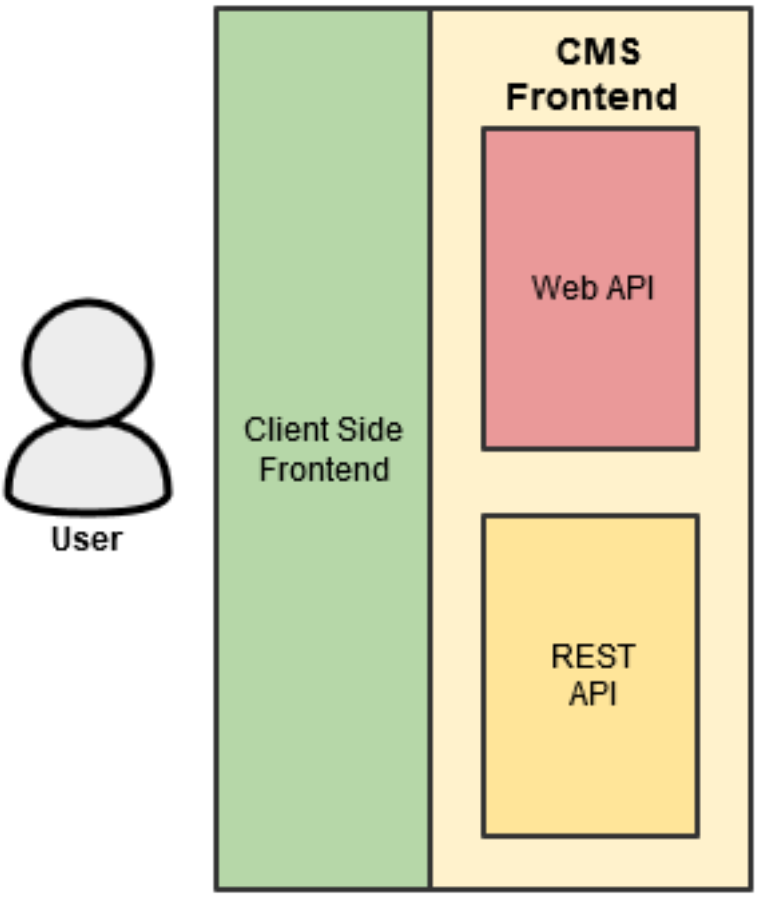

Figure 6.2: The sub-modules of the CMS Frontend.

The frontend itself is divided into two sub modules - these are the client side frontend and the server side frontend (or just CMS frontend). These can be seen in the Figure

6.2. As can be clearly seen the client-side frontend faces the user.

The client side frontend is where most of the cryptographic methods are to reside. To allow users to generate certificates and backup and recover keys, client side cryptographic methods are needed. Apart from this, the client side sub module takes user data, processes it and forwards it to the server side frontend for further processing.

The server side frontend, takes care of processing the data received from the client side frontend in to a format that can be sent over the network. It maps client side requests with appropriate backend functions and gets back the response from the backend and forwards it to the client side frontend for presentation to the user.

6.2.2 CMS Backend

The back-end provides for all server side functionalities that process user requests made to the front-end of the service. This component also interacts with external components, namely the database storage for the service and also the Registration Authority or the Certifying authority (depending upon the hierarchy of the Public Key Infrastructure). The interaction with the RA or the CA, would need to be done using API calls. This component also takes care of all Database related interactions.

The backend also provides an interface that a directory service can interact with primarily to fetch data like issued certificates or signed OpenPGP keys from the database..

The design of the CMS backend constitutes of five sub-modules as is seen in Figure 6.3. These sub-modules carry out specific tasks. The sub-modules are as follows:

1. CA Module - This module takes care of all functionalities and interactions related to the CA. The module receives requests meant to be forwarded to a CA or RA. It processes these requests and uses appropriate API calls or other methods of the communication supported by the CA, to forward the requests to the CA. It then also processes the response from the CA and makes the response available for the certificate management service. For example, the CA module could receive a certification request and then forward it to the CA. It is then to get an issued certificate from the CA (provided all checks by the CA on the certification request pass). The issued certificates can then also be stored in the database of the service by this module.
2. OpenPGP Module - This module is to take care of all OpenPGP related functionalities and helps in OpenPGP certificate processing. It primarily concerns itself with the signing of received OpenPGP keys with the certificate management service's key. It can then store the signed key in the database of the service.
3. Database Access Module - This module mainly concerns itself with database access and encapsulating the database attached to the certificate management

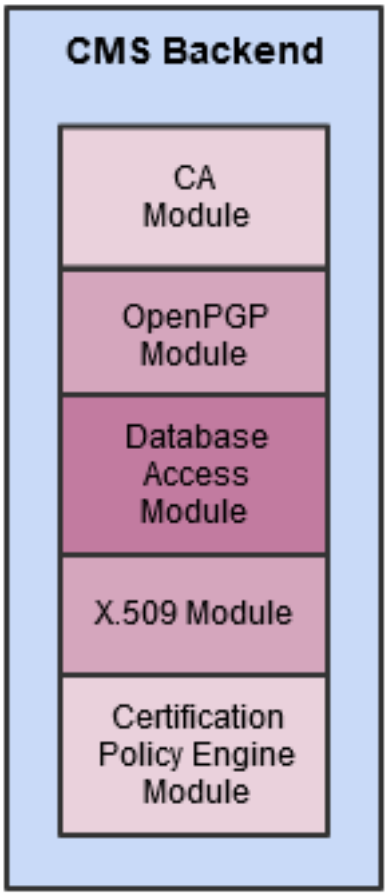

Figure 6.3: The sub-modules of the CMS Backend.

service. All requests sending data for storage to the database or fetching data from the database will have to interact with this layer.

4. X.509 Module - This module focuses on carrying out X.509 related functionalities and X.509 certificate processing. All X.509 related tasks are performed by this module. Most of these functions will be to extract information from X.509 certificates or to format X.509 certificates from one format to another.

5. Certificate Policy Engine Module - This module will concern itself with checking incoming certificate signing requests. It will check these requests against a certificate policy that has been established by the service operators. A certificate policy is a document which aims to state what are the different actors of a public key infrastructure, their roles and their duties. It is a method for CAs to implement some level of control and syntactic or semantic checks on certificate signing requests being received by it. It enforces users to follow CA guidelines and rules by

acting as a gatekeeper between the CA and the end users. If a certificate signing request fails this check, then the request will be automatically declined.

6.2.3 Interfacing

As per the proposed functionalities and other requirements, the best way to realise a part of the interfacing in the certificate management service would be to create a RESTful web application. This interfacing is primarily concerned in providing a REST API for external access to the service and to interface between the client side frontend and the server side frontend. REST stands for Representational State Transfer. It is an architecture style for designing networked applications. It relies on a stateless, client-server, cacheable communications protocol. In the case of the certificate management service, this protocol is to be HTTP. Simple HTTP calls can be used to make requests between machines over the network.

6.3 Cryptography and Encryption

Cryptographic functions need to be available in the front-end as well as in the back-end. As seen in the requirement analysis, the end user can generate his or her own certificate signing request. This can only happen, if sufficient cryptographic functionalities are provided for in the frontend itself. This needs to be realised by allowing for generation of public-private key pairs and a corresponding certificate signing request in the client browser (Given that this will be how an end user will interact with the service's frontend). The design of these cryptographic mechanisms needs to follow current standards and needs to be robust. It has to also see to it that user interaction is limited and when needed, it is easy for the user to understand the tasks he or she is to perform. This would help increase the usability of the service from a security viewpoint. As discussed earlier, cryptography creates an impediment for many a normal users. Hence, the design needs to take this into consideration.

Another requirement of the service is to provide for secure backing-up and recovery of private keys. This implies that the private key of the user can be securely encrypted in the frontend and then transmitted to the backend for storage. When it comes to recovery, the encrypted key needs to be transmitted from the backend storage to the frontend and then decrypted on the client-side. Once again secure and robust cryptographic methods need to be implemented in the frontend to allow for this to be possible. Furthermore, if a user wishes to backup his or her key at a later stage, the frontend should provide for public-private key verification so as to make sure that the private key being uploaded matches the corresponding public key for a certificate signing request.

The cryptographic methods in the backend would basically be used to interact with

certificate signing requests and OpenPGP keys. An example for certificate interaction is to extract the public key from a certificate signing request or to help sign uploaded OpenPGP keys. Similar interaction methods are available in the backend.

Hence, these considerations would have to be taken into account while designing the cryptographic functions in both the frontend as well as the backend.

6.4 Database Storage

The backend requires a database to store all service-specific data. Various requests would be created by users using the fronend. These requests could also be of various types. Hence a database would need to be designed that can sufficiently store all this data. This would include certificate signing requests to revocation requests. A user's private keys also would be securely stored in this very database.

A database encapsulation layer would be needed so that the backend logic of the service does not have to worry about the implementation details of the database (including the physical schema of the database). This layer would provide the needed business objects with persistence services (i.e. the ability to read, write and delete data from data sources). This encapsulation would provide various benefits for the certificate management service primarily by removing the coupling between the object schema and data schema of the service. Hence, either one of these schemas can be further developed without affecting the other one or causing issues between their interactions with one another. This would also make it easier to implement the service, as all database related implementations will be in one place.

6.5 PKI Infrastructure

The certificate management service does not incorporate its own CA. Hence, it needs to be designed such that it can interact with an external CA. Therefore, the design needs to consider the needed PKI infrastructure of RA(s) and CA(s). The best possible design approach for this would be to design an easy to use API or Application Programming Interface, that can access a Public Key Infrastructure. For this thesis, either of the earlier discussed systems can be used - OpenCA, Dogtag or EJBCA. These systems can help set up a CA that the service should be able to interact with to complete its own tasks. This will be done through the API. The most important of these tasks is to forward a certificate signing request to a CA so as to get the CA to issue a corresponding certificate for that certificate signing request. Apart from this requests to revoke certificates, renew certificates, etc. also need to be forwarded by the service to the CA. Lastly, this API would help in making the service more extensible, as in the future, new functionalities

can be added so that the service can be extended interact with another kind of CA system.

Chapter 7

Implementation

This chapter discusses the implementation of the web service. The previous chapter has laid down the design for the implementation. This chapter discusses how the design considerations from the previous chapter were implemented to create the certificate management service.

7.1 Building Blocks of the Service

This section discusses the frameworks, libraries and packages that were used to implement the design for the certificate management service. The frameworks were used so as to meet our requirements of a robust, extensible system that follows a RESTful architecture. The certificate management service was implemented using the Spring Framework.

7.1.1 Spring Framework

The Spring Framework [33] is an open source application framework and inversion of reference container for the Java platform. The core features of the Spring framework can be used by any Java application. There are also extensions available for building web applications on top of the Java EE platform. Spring Framework allows for the creation of high performing, easily testable, reusable code. The certificate management service relies on the Spring MVC (Model view container) framework. It provides a model-view-controller architecture and ready components which can be used to develop a loosely coupled as well as a flexible web application. The whole Model View Controller pattern results in the separation of different aspects of the application - the input logic, business logic and the User interface logic. At the same time, these aspects are all loosely coupled.

7.1.2 Spring Security

The certificate management service also uses an extension of the Spring framework called Spring Security. Spring Security is a powerful and highly customizable authentication and access-control framework. This framework is used to help with securing the certificate management service that is to be designed using the Spring framework. Spring security provides comprehensive and extensible support for both authentication and authorization of users. It also protects against various attacks that can occur on applications like session fixation, cross site request forgery, etc.

7.1.3 Spring Data JPA

Another module of Spring Framework that is used by the certificate management service is the Spring Data JPA (Java Persistence API) module which deals with enhanced support for JPA based data access layers. The Java Persistence API (JPA) is a Java specification for accessing, persisting, and managing data between Java objects or classes and a relational database. This module helps in cutting down on the amount of work required to create a data access layer for an application. This module allows the backend of the certificate management service to easily interact with the database.

Next, the various libraries and packages used to implement the certificate management service are discussed. The libraries are, namely, Web Cryptography API, PKI.js and Bouncy Castle.

7.1.4 Web Cryptography API

Web Cryptography API or WebCrypto is a JavaScript API for performing basic cryptographic operations in web applications. This API provides for numerous cryptographic operations like hashing, encryption, decryption, generation of keys and so on. Uses for this API range from user or service authentication, document or code signing, to the confidentiality and integrity of communications. It defines a low-level interface for interacting with cryptographic key material that is managed or exposed by client side browsers. This API allows applications to perform operations such as signature generation and verification, hashing and verification, encryption and decryption, without requiring access to the raw keying material. This API is used by the frontend of the application for generating Public-Private key-pairs, encryption keys, encrypting data and decrypting data.

7.1.5 PKI.js

PKI.js [34] is a pure JavaScript library implementing the formats that are used in PKI applications (signing, encryption, certificate requests, OCSP and TSP requests/responses). It has been built and maintained by GlobalSign and Peculiar Ventures. It is built on WebCrypto (Web Cryptography API) and requires no additional plug-ins. This library provides for creation of Certificate Signing requests in the browser and is used especially for this purpose in the frontend of the certificate management service

7.1.6 Bouncy Castle

Bouncy Castle is a collection of APIs and libraries used in programming cryptography-related tasks. Bouncy Castle includes APIs and libraries for the Java programming language as well as the C programming language. Bouncy Castle was developed and is supported by an Australian non-profit organization called Legion of the Bouncy Castle. The Java package of Bouncy Castle is organised such that it contains a light-weight API suitable for use in any environment with the additional infrastructure to conform the cryptographic algorithms to the Java Cryptography Extension or JCE framework. Bouncy Castle supports X.509 certificates, S/MIME, Certificate Management Protocol or CMP and OpenPGP. It is a perfect bundle for the certificate management service that needs to be implemented. The Bouncy Castle package is used only in the backend to carry out server side cryptography operations.

All of the above discussed frameworks, libraries and packages were used to fully implement the design of the certificate management service. They were chosen as the best options to proceed with implementing the design of the service.

7.2 CMS

The certificate management service or CMS, was implemented by dividing it into two modules - the CMS Frontend and the CMS Backend.

Figure 7.1, shows an abstract component diagram for the implemented certificate management service. As can be seen, the service has been divided into main modules - the frontend and the backend. The client side frontend is also a part of the frontend, and is only shown separately in the picture to highlight how the frontend functions. A user sees the graphical user interface of the certificate management service via the client side frontend. The CMS Frontend has two modules as shown in the figure - the Web API and the REST API. The CMS Frontend receives inputs through REST API calls and responds with HTTP responses.

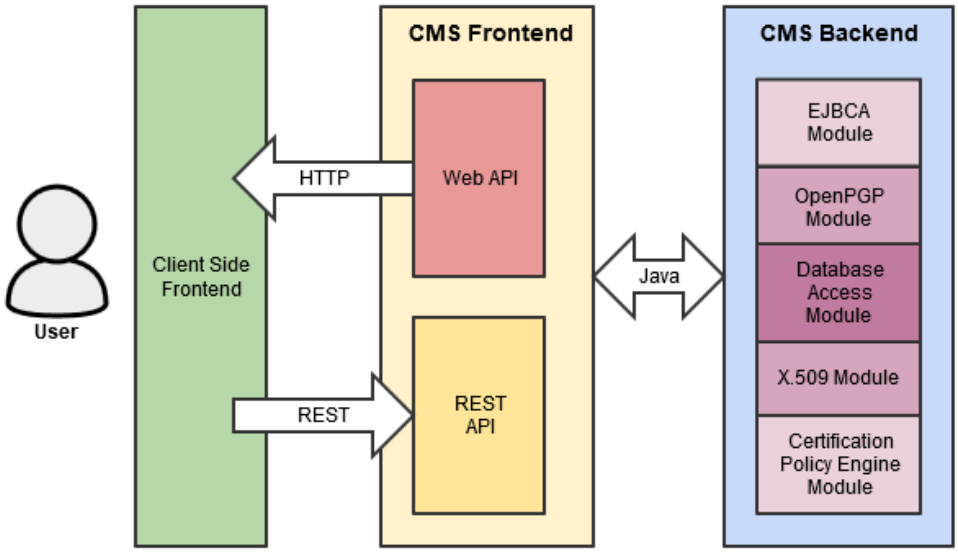

Figure 7.1: An abstract component diagram for the implemented certificate management service.

The backend and the frontend interact using Java. The backend in turn has various sub-modules as already discussed in the design chapter. These can be clearly seen in the figure.

7.2.1 CMS Frontend

The CMS Frontend uses the Spring MVC Framework. Figure 7.2 shows the packaging of the Frontend.

Application.java is the entry point for the application. It helps start the Spring Application. The SecurityConfig.java is a Spring Security specific class that takes care of Security related tasks. It configures the application to prevent unauthenticated access to the various pages of the service. The SecurityConfig.java class also holds dummy authentication credentials for three kinds of users. These credentials can be used to access the application in the absence of an external Authentication component. These are as follows (these are presented in the format of username:password:role):

1. End User - user:user:USER
2. Support Staff - support:support:SUPPORT
3. Service Operators - service:service:SERVICEOPERATOR

Since, an external authentication component is unavailable, SecurityConfig.java helps implement a similar component for the needs of implementing the service. Apart from

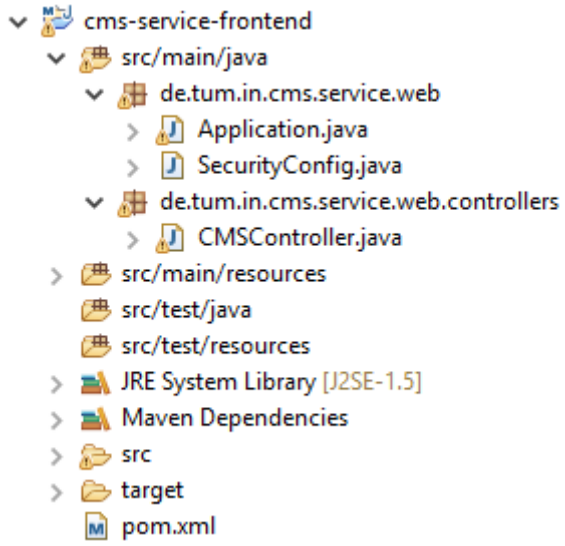

Figure 7.2: The packaging of the main classes in the Frontend.

the dummy credentials, SecurityConfig.java also authenticates users using username, password and role data from the database of the service. It also takes care of user registration.

The CMSController.java class is the primary controller of the frontend of the certficate management service. It holds all the Request Mappings that are used to map web requests to Spring Controller methods as is the case in Spring MVC. This controller interacts with the backend by forwarding the requests to the backend. It forwards these requests to the CSRManager.java class in the backend. It also gets the responses from the backend and then forwards it to the user.

The rest of the frontend consists of resources needed to make the web application usable. Templates were used to create the various pages and other HTML constructs (like the menu bar) for the application. These are coded in HTML and Javascript. Figure 7.3 shows all the templates that are stored under resources in the frontend. They form the Graphical User Interface of the application. The stylesheet of the application along with Javascript libraries (including the PKI.js library) are stored under the static folder while all the HTML code is stored under the templates folder. Most of these pages have inline Javascript to create requests that are mapped by the CMSController class. Javascript is also used to process the responses sent by the backend and then forwarded by the CMSController class.

The templates were divided on the basis of the roles of the users. Each function that a user can perform has its own template.

The basic common templates are as follows:

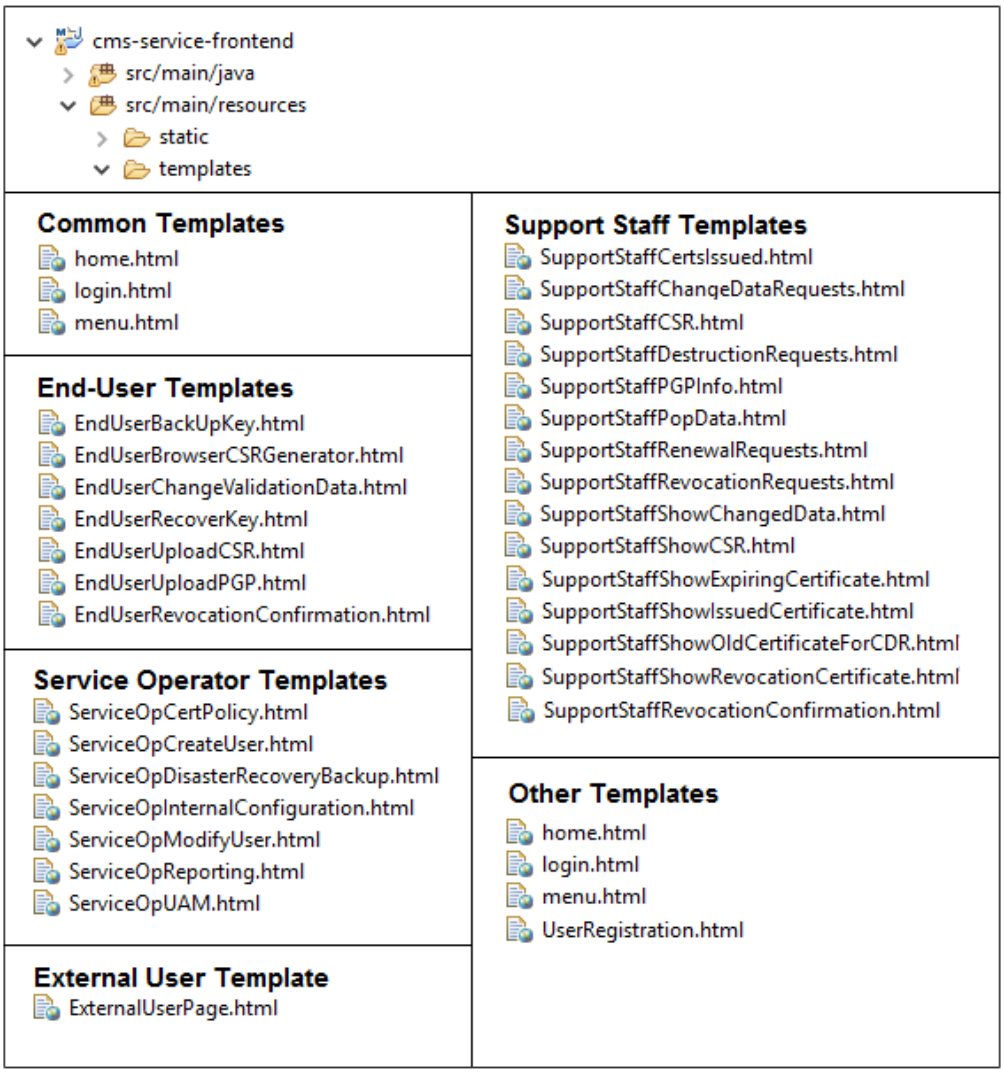

Figure 7.3: The templates in the frontend are stored under resources.

1. home.html - This is the home page of the application. After login, the data shown on this home page varies from user to user.
2. login.html - This is the login page. Spring Security Configuration that had been done in the SecurityConfig.java class makes sure that this is the landing page of the application and no user can access any other page without authenticating himself or herself first.
3. menu.html - This is the HTML code which is included in all pages and populates the navigation bar for the application. Navigation options change depending on the role of the user. Once again here Spring Security comes into help.
4. UserRegistration.html - This is the page where a user can register himself or

herself with the service. It works in conjunction with the SecurityConfig.java.

Next are the templates for the End User:

1. EndUserBackUpKey.html - This form helps the user back up his or her own private key to the server side storage at any point of time.

2. EndUserBrowserCSRGenerator.html - This page helps generate a public-private key pair for a user and a corresponding CSR, that can then be uploaded to the backend Server for approval. The page also allows for secure backup of the private key to the server.

3. EndUserChangeValidationData.html - This page allows a user to create a Change Data Request wherein he or she can request for change of some validation data in a certificate that has already been issued. This is very similar to the functions provided in the EndUserBrowserCSRGenerator.html template. This is because, to change certificate data, the previously issued certificates need to be revoked, and a new certificate signing request needs to be generated with the changed data.

4. EndUserRecoverKey.html - This page allows the user to recover a private key for a certificate or certificate signing request provided that the key was backed-up

5. EndUserUploadCSR.html - This page allows the user to upload a certificate signing request to the backend server for approval. This CSR could have been generated by the user using any external tool like the OpenSSL command line tool.

6. EndUserUploadPGP.html - This page allows a user to upload his or her OpenPGP key to the server for being signed by the Server's own OpenPGP key (provided it is approved by a Support Staff).

7. EndUserRevocationConfirmed.html - When a user decides to revoke his or her certificate from the home page, this template pops-up. Here a user enters the description of the reason for which he or she wishes to revoke their certificate.

The following are the templates for the Service Operators:

1. ServiceOpUAM.html - This page allows the service operators to carry out all User Access Management related tasks. It shows all available users to the operators. Operators can Create new users, Modify the data about users or Delete Users through this page.

2. ServiceOpCreateUser.html - This page is complementary to the ServiceOpUAM.html and allows for creation of new users.

3. ServiceOpModifyUser.html - This page is complementary to the ServiceOpUAM.html and allows for modification of user data.

4. ServiceOpInternalConfiguration.html - This page allows service operators to configure the web application internally.
5. ServiceOpDisasterRecoveryBackup.html - This page helps in tasks of back-up and disaster recovery of the web application.
6. ServiceOpReporting.html - This page allows service operators to audit the web service and check the logs of the web application.
7. ServiceOpCertPolicy.html - This page allows service operators to set up an application wide certification policy. This certification policy has to be adhered to by all users and all certificate signing requests are to follow it.

Lastly, the following are the templates for the Support Staff:

1. SupportStaffCertsIssued.html - This page shows all the certificates that have already been issued using the certificate management service. It also shows corresponding user details in a tabular form. The Support Staff can view each certificate in a new window.
2. SupportStaffShowIssuedCertificate.html - This page is complementary to the SupportStaffCertsIssued.html page. It can be used by the support staff to view individual certificates.
3. SupportStaffChangeDataRequests.html - This page shows all change data requests submitted to the service in a tabular form. The Support Staff can approve or decline a change data request. They can also view the new data in a new window as well as the current certificate in another window. If approved, they can send the request to the CA for approval and issuance of a new certificate with the new data.
4. SupportStaffShowChangedData.html - This page is complementary to the SupportStaffChangeDataRequests.html page. It shows the data that has been requested to be changed.
5. SupportStaffShowOldCertificateForCDR.html - This page is complementary to the SupportStaffChangeDataRequests.html page. It shows the certificate as it was originally issued and for which a change data request has been made.
6. SupportStaffCSR.html - This page shows all the Certificate Signing Requests or CSRs that have been received by the certificate management service in a tabular form. The Support Staff can approve or decline a CSR. They can also view each CSR in a new window. If approved, they can send the CSR to the CA for approval and issuance of the certificate.
7. SupportStaffShowCSR.html - This page is complementary to the page mentioned in the point above. It shows a Certificate Signing Request to the support staff.

8. SupportStaffRenewalRequests.html - This page shows all the Certificate Renewal Requests that have been received by the certificate management service in a tabular form. The Support Staff can approve or decline a request. They can also view each expiring certificate in a new window. If approved, they can send the request to the CA for approval and issuance of a renewed certificate.

9. SupportStaffShowExpiringCertificate.html - This page is complementary to the SupportStaffRenewalRequests.html page. It shows the expiring certificate for which renewal is requested.

10. SupportStaffRevocationRequests.html - This page shows all the Revocation Requests that have been received by the certificate management service in a tabular form. The Support Staff can approve or decline a request. They can also view the certificate to be revoked in a new window. If approved, they can send the request to the CA for revocation of the certificate.

11. SupportStaffShowRevocationCertificate.html - This page is complementary to the SupportStaffRevocationRequests.html page. It shows the certificate that needs to be revoked for which the revocation request has or had been made.

12. SupportStaffDestructionRequests.html - This page shows all the Certificate Destruction Requests that have been received by the certificate management service in a tabular form.

13. SupportStaffPGPInfo.html - This page shows all the OpenPGP Key Signing Requests that have been received by the certificate management service in a tabular form. The Support Staff can approve or decline a request. If approved, they can sign the corresponding OpenPGP key in a request with the server's own key.

14. SupportStaffPopData.html - This page was primarily used to populate the database of the service of dummy data for testing. It is not part of the final implementation. It fills up the database with sample certificates and certificate signing requests and creates new requests in each table. The page can also be used to delete all existing data in the database.

15. SupportStaffRevocationConfirmation - When approving a user's revocation request on the SupportStaffRevocationRequests.html page, this page pops-up. The Support Staff can then view the description provided by the user describing why he or she wishes to revoke their certificate. The support staff can then chose an appropriate revocation code and approve the request.

7.2.2 CMS Backend

Figure 7.4 shows the packaging of the Backend.

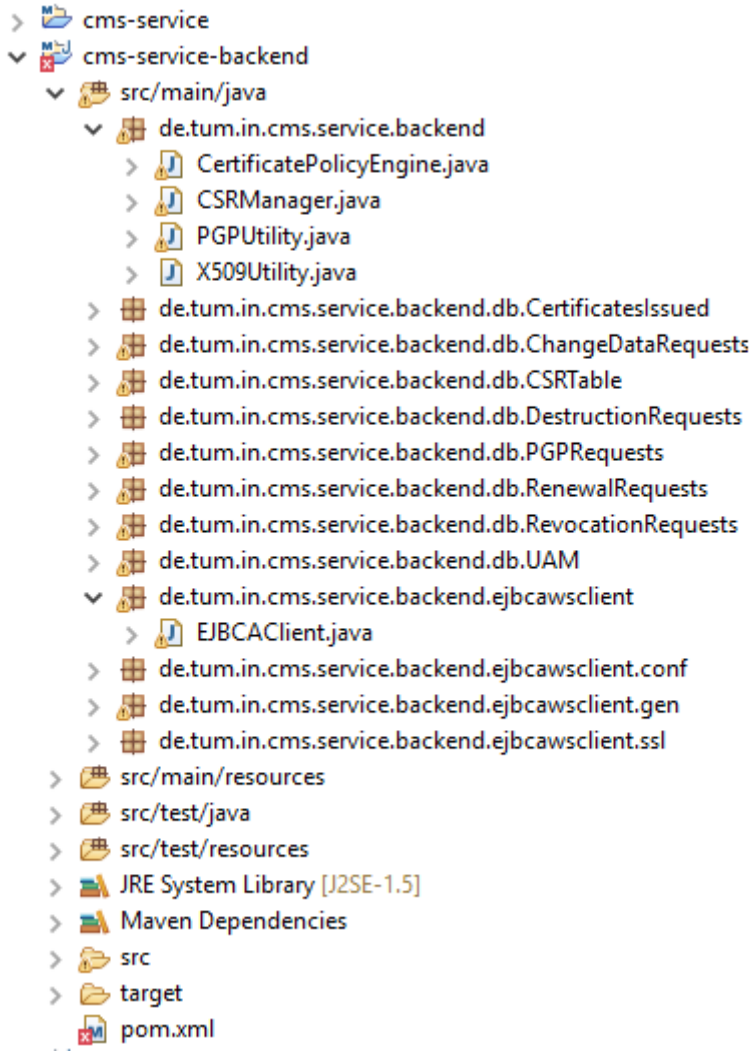

Figure 7.4: The packaging of the main classes in the Backend

CSRManager.java is the main class in the backend of the certificate management service. It receives requests from the frontend of the service (through the Controller class there) and processes these requests. Once processed it sends back the responses to the frontend. This class holds primarily methods to interact with the Database Access Module and to carry out some string formatting operations.

CertificatePolicyEngine.java is a class that checks incoming certificate signing requests to be compliant with the Certificate Policy set up by service operators. As the name suggests, this class implements the Certificate Policy Engine module that was put forth in the design chapter.

PGPUtility.java provides utility functions which help in signing OpenPGP keys. While the signing takes place in CSRManager.java, some functions needed for that whole

process are provided for by the PGPUtility.java. This is the implementation of the OpenPGP Module.

X509Utility.java class provides utility functions to work with X.509 certificates. It implements the X.509 module from the design.

EJBCAClient.java class provides the fucntionality to interact with the external CA (implemented using EJBCA). It is the implementation of the CA module from the design.

The other packages in the back-end are all related to the Data Access Layer and are explained in the next section.

7.3 Database

An Oracle database was created to be used by the certificate management service. Each table, except the `CMS_USERS` table, has a STATUS column that is used widely to check status of requests.

The following tables were created:

1. `CERTIFICATES_ISSUED` - Once, the certificates have been issued by a CA, they are stored i;n this table along with other details of the user the certificate belongs to.
2. `CHANGE_DATA_REQUESTS` - This table holds all the requests for changing the data in a certificate.
3. `CSR_TABLE` - This table holds the certificate signing requests, as well as the private key (if backed-up).
4. `DESTRUCTION_REQUESTS` - Any certification destruction requests are stored in this table.
5. `PGP_TABLE` - All requests to sign OpenPGP keys and the subsequently signed OpenPGP keys are stored in this table.
6. `RENEWAL_REQUESTS` - Certificate renewal requests are stored in this table.
7. `REVOCATION_REQUESTS` - Requests for revoking certificates are stored in this table.
8. `CMS_USERS` - This table is used for the purpose of User Access Management for the service itself.

The figure 7.5 shows the packaging of the various classes used to implement the Data Access Layer for all the previously mentioned tables in the database that is being used by the certificate management service. These classes collectively implement the Database Access Module from the design. Each table has been provided with an Entity class, a corresponding repository and a corresponding service class.This created the data access

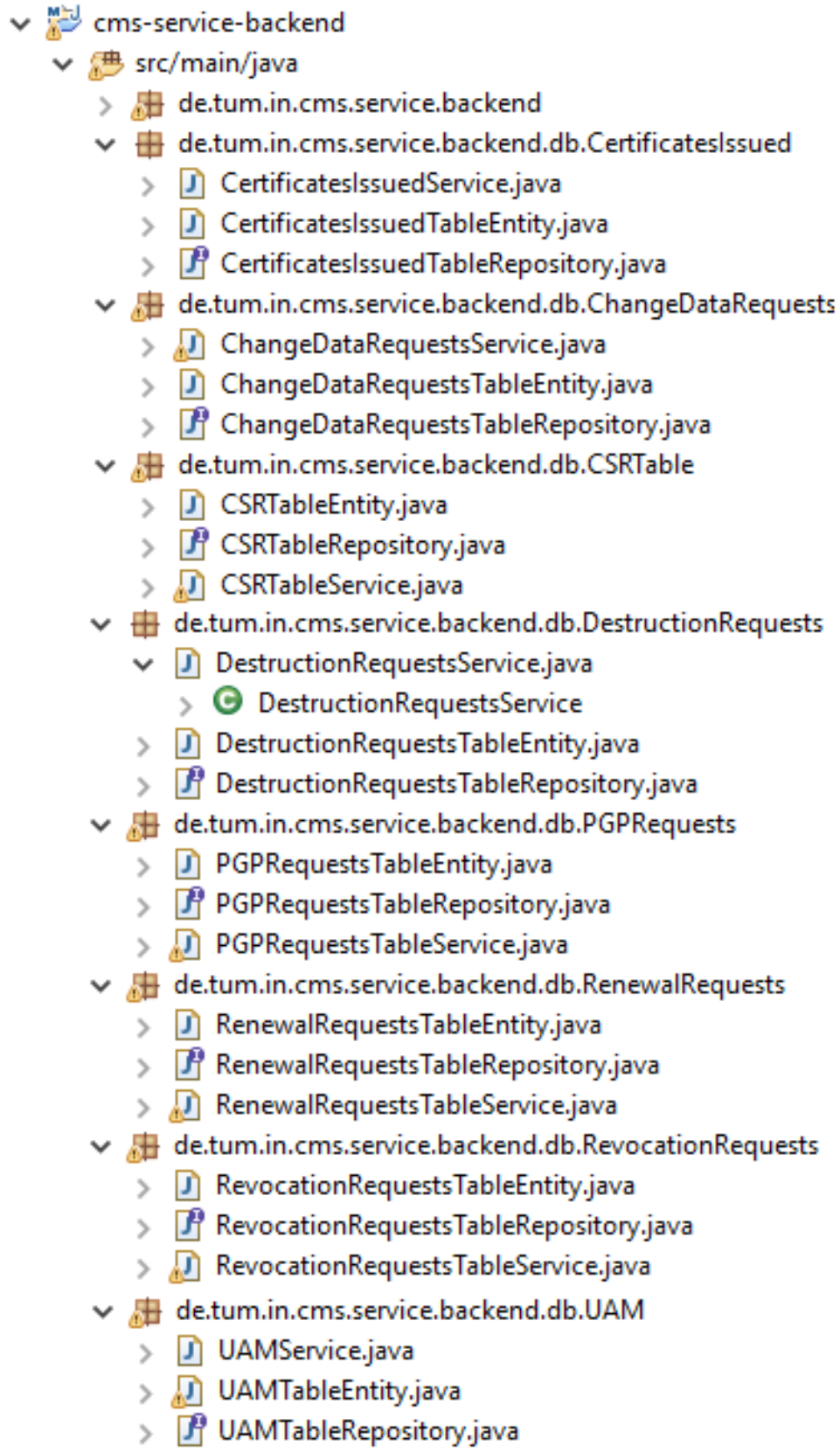

Figure 7.5: This is the structure of the packaging for the classes used to implement the Data Access Layer for the service.

layer for the certificate management service. This was done on the backend side of the service. A Repository represents all objects of a certain type as a conceptual set. It acts like a collection, except with more elaborate querying capability. The entity class represents the table itself while the service class is where all the query methods are coded. This was done with the help of Spring Data JPA.

7.4 PKI

While implementing a suitable PKI for the certificate management service to interact with, certain problems were faced.

Initially, OpenCA was planned to be used to implement the needed PKI. This was mostly to set up the CA that the service would interact with. However, setting up OpenCA proved to be a major impediment. The official installation guides from OpenCA creators were unavailable as the wiki on their page for the same was down. A user guide created by the developers of OpenCA was available but the Installation guide supposedly in their wiki was unavailable. Tutorials on the Internet created by users, though outdated, were next used to implement the PKI using OpenCA. Significant progress was made in this area until some errors emerged after the setup was finished, and the OpenCA GUI was tried to be opened. Troubleshooting help for the same was unavailable. The official documentation could have been used to probably rectify the issue but was unavailable. Another document created by OpenCA Research Labs as a guide to OpenCA was informative but was not a definitive installation and troubleshooting guide. Owing to this, OpenCA was discarded as the preferred implementation for the PKI component for the certificate management service.

Upon further investigation of OpenCA, it was also found out that OpenCA does not provide any kind of service or API, which the certificate management service could interact with over a network. OpenCA offers a well made Graphical User Interface that can carry out all the provided CA operations but this does not meet the requirements of the certificate management service. Hence, the conclusion was reached that even if OpenCA would have functioned correctly, it would have not been the right choice as the PKI component for the certificate management service (as far as the scope of this thesis is concerned).

Finally, after investigating other CA systems available for free on the Internet, EJBCA was chosen as the preferred choice. EJBCA allows for external applications and services to interact with it using the Certificate Management Protocol. EJBCA website also provides for a Virtual Machine with the EJBCA set up on it. This was thus used to implement the PKI component of the design for the certificate management service.

7.5 Processing Certificates and Requests

This section discusses how the underlying processes of the certificate management service were implemented.

Many processes central to the certificate management service required the use of cryptographic methods. Cryptography was implemented in the frontend through javascript. As mentioned earlier the PKI.js Javascript library was used in the frontend along with the Web Cryptography API. All cryptographic tasks thus were implemented using Javascript. In the backend, cryptographic functions are implemented in the CSRManager.java class, PGPUtility.java class, X509Utility.java class as well as the EJBCAClient.java class.

The following sections discuss the processes involved in processing certificates and requests.

7.5.1 Generation of a Certificate Signing Request

The generation of a certificate signing request begins in the EndUserBrowserCSRGenerator.html template. Here the end user can generate a certificate signing request in the Browser itself. Using PKI.js and Web Cryptography API, a key pair is initially generated. After this a CSR is created. The user can then chose how he or she wishes to transmit the CSR to the server - either with a private key backup or without. In case, the user chooses to backup his or her key, methods exist to help encrypt the key before being sent to the server. This method first generates a random AES key and then uses it to encrypt the private key of the user. This AES Key is given to the user so that it can be used to decrypt the encrypted private key when needed (in the future). A prompt asks the user to re-enter the secure code, to make sure the user has copied the secure code somewhere safe. Only after the user re-enters the exact same generated AES key, does the encryption of the private key begin. Once this finishes, the key is sent to the server for storage.

Figure 7.6 shows the sequence diagram of the process of generation of certificate signing requests. In this sequence diagram, it is a given that a user submits his or her browser-generated CSR to the server along with a secure back-up of his or her private key. As can be seen, once the private key has been encrypted, the CSR along with the public and private key are sent as part of the request to the backend. Here the CSR is forwarded to the Certificate Policy Engine Module to carry out semantic checks on the CSR. If these checks pass, the CSR is then stored in the database of the certificate management service.

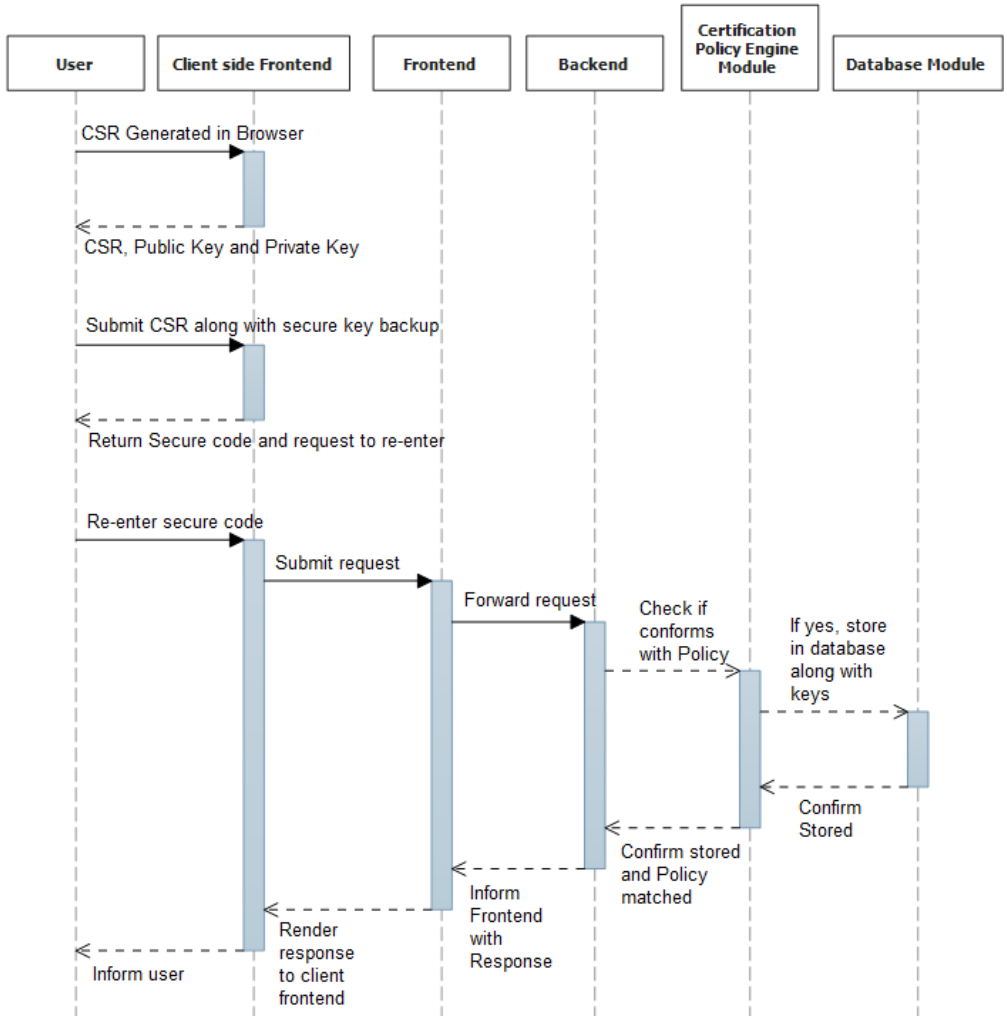

Figure 7.6: Sequence diagram for the process of generating a CSR and submitting it along with a secure back-up of the private key.

7.5.2 Issuing Certificates

A support staff user can use the SupportStaffCSR.html template to approve recieved CSR requests. The support staff user basically approves a particular CSR and then sends it to the CA for certificate issue. When this request is made by the Support Staff user, the backend initially fetches the CSR and other user details related to that CSR from the database. It then provides all this information to the EJBCA Module. The EJBCA module uses this information to interact with the CA (implemented using EJBCA). It makes a Certificate Request to the CA, which then results in a response from the CA which has the issued certificate. This certificate can then be stored in the database and distributed to the user. Figure 7.7 shows a sequence diagram for this process.

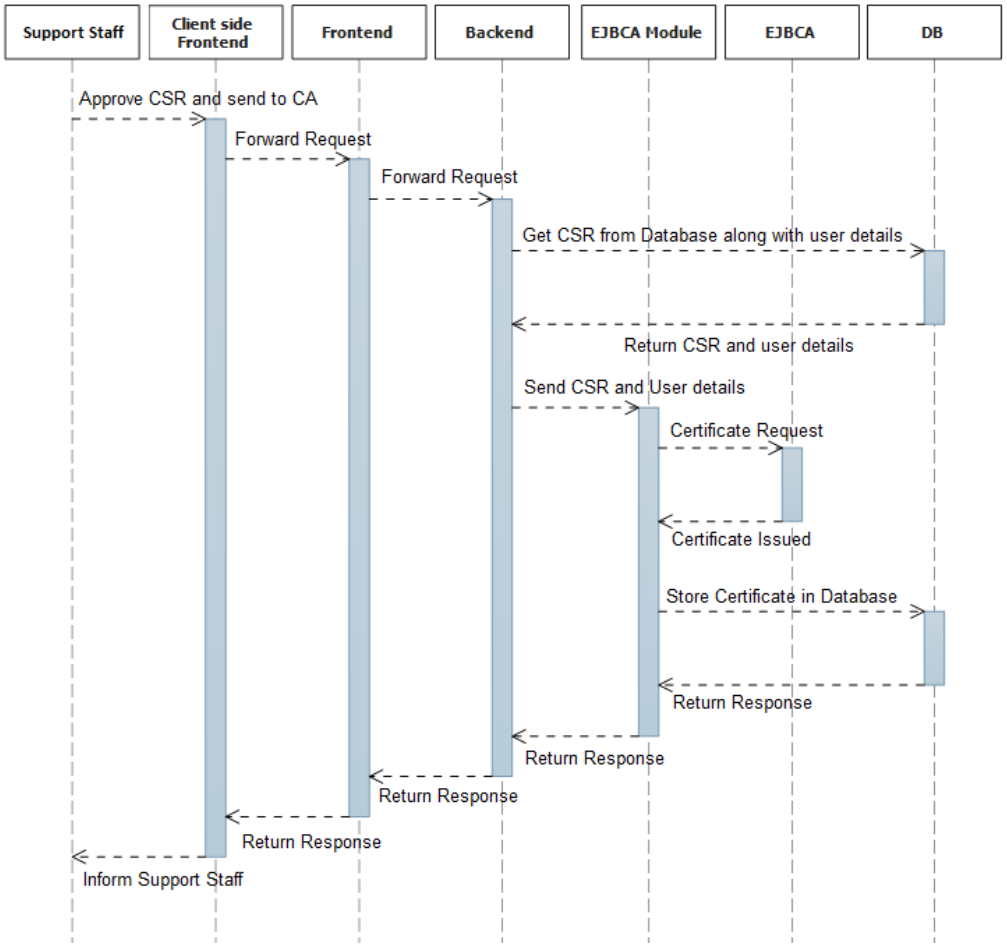

Figure 7.7: Sequence diagram depicting the process of a certificate being issued by the CA once a CSR is approved by a Support Staff user.

7.5.3 Private Key Backup

The back-up of a private key happens in the EndUserBackUpKey.html template. Although, the EndUserBrowserCSRGenerator.html template provides secure key backup functionality too, the EndUserBackUpKey.html template allows users to backup their keys at a later moment. These could also be keys for certificate signing requests that the user may have uploaded to the certificate management service without using the Browser CSR generator. The cryptographic functions on this page check whether the private key corresponds to a CSR or certificate already stored in the service's database. The user can select which CSR or Certificate the private key belongs to before uploading. Next, the verification to see if the private key corresponds to the public key is done on the client side. Following this, the user can chose whether he or she wishes to store the key on the server in an encrypted form or in plain-text. Therefore, if the user wishes, methods exist in this template to allow the user to encrypt the private key with a ran-

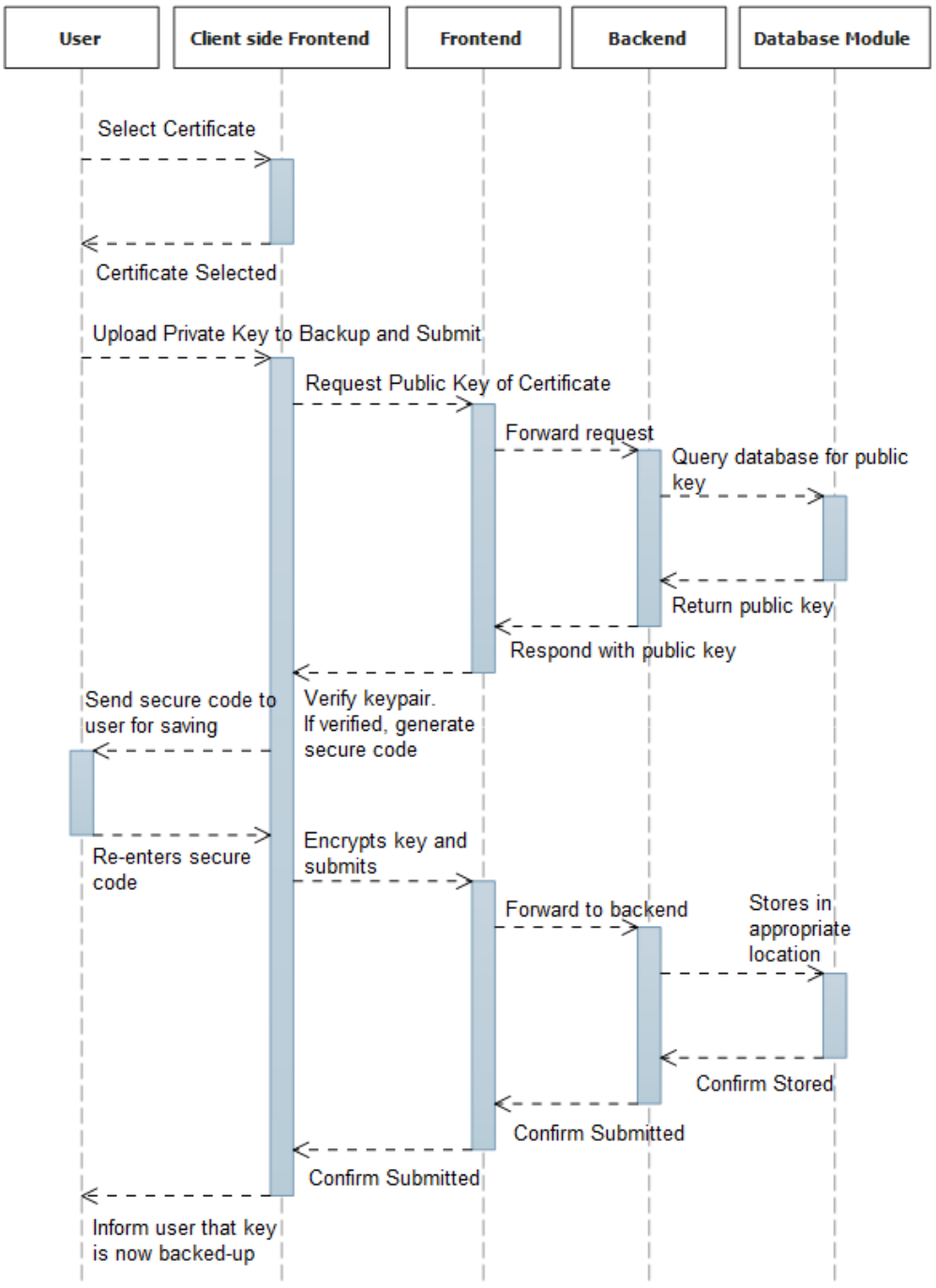

Figure 7.8: Sequence diagram depicting the process of backing-up a key securely to the service.

domly generated AES key (as in the case of the EndUserBrowserCSRGenerator.html template) before the private key is transmitted to the server for secure storage. Figure 7.8 shows the Sequence diagram detailing the process of key back-up. Here its a given that the user wants to store his or her private key securely with the certificate management

service. The process requires the user to first select the appropriate certificate to which the key belongs. The service then fetches the public key of the certificate. Once fetched, on the client side, a random string of data is first signed using the fetched public key. Then this signed data is verified using the private key that the user is trying to back-up. If the verification passes, it means that the private key belongs to the selected certificate. In this case, the back-up of the key can proceed. The key is encrypted in a similar fashion as to how it is encrypted in the EndUserBrowserCSRGenerator.html template. AES encryption is used here and a random AES Key or secure code is presented to the user. This code is used to encrypt the private key and it is then transmitted to the service backend for storage in the database.

7.5.4 Signing OpenPGP Keys

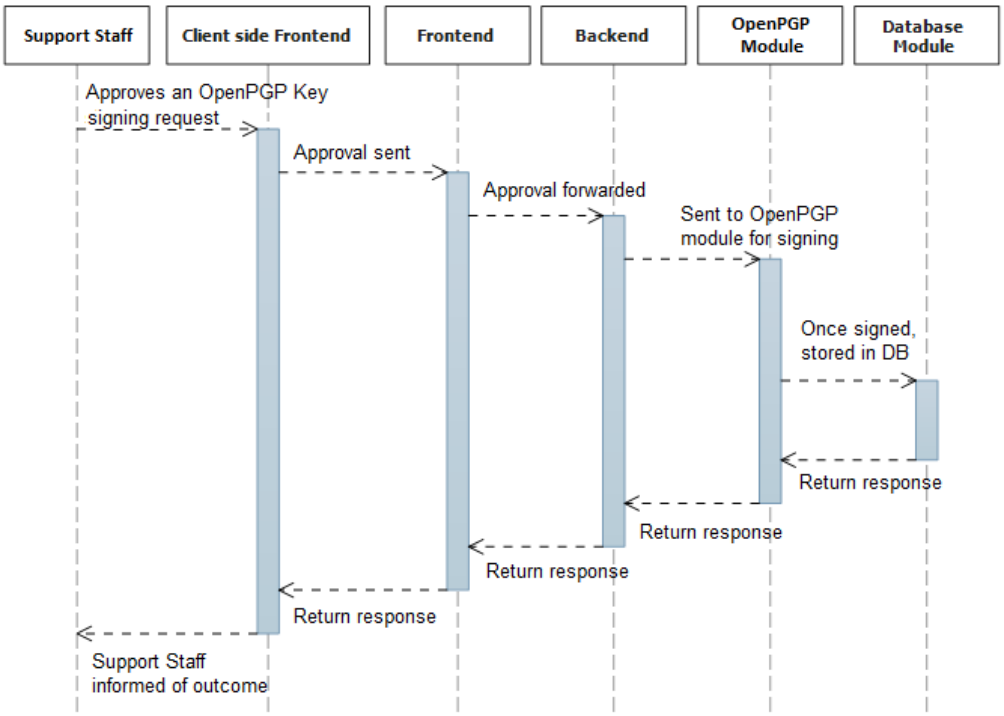

Figure 7.9: Sequence diagram depicting the process of signing of OpenPGP keys once a key singing request has been approved by a Support Staff user.

There is no particular template for signing OpenPGP keys. An end user can upload his or her OpenPGP key using the EndUserUploadPGP.html template. This will create a key signing request which the Support Staff can see in the SupportStaffPGPInfo.html template. The Support Staff can then approve a request and the key submitted by the user is sent to the PGP Module. Here the key is signed using the key provided in the internal configuration of the certificate management service. The signed key is then added to the database. Figure 7.9 shows the sequence diagram for this process. A similar

design is followed for other processes which require approval of the support staff - like renewal requests and revocation requests.

7.5.5 Recovering Private Keys

Recovery of Keys is carried out in the EndUserRecoverKey.html template. Here the end user can recover a private key (if backed up to the server). The end user needs to first select which certificate or CSR he or she wish to recover the key for. Once selected, they can click the Recover button. The encrypted key is then transmitted from the backend to the frontend and a decryption process begins. The end user will have to enter the same secure code he or she were presented with while backing-up this private key or else the key will not be decrypted. A sequence diagram for the key recovery process can be seen in Figure 7.10. As can be seen, once the user requests for the recovery of a key, initially the back up status of the key is checked with the database. Only if the key is present in the database, does the process continue.

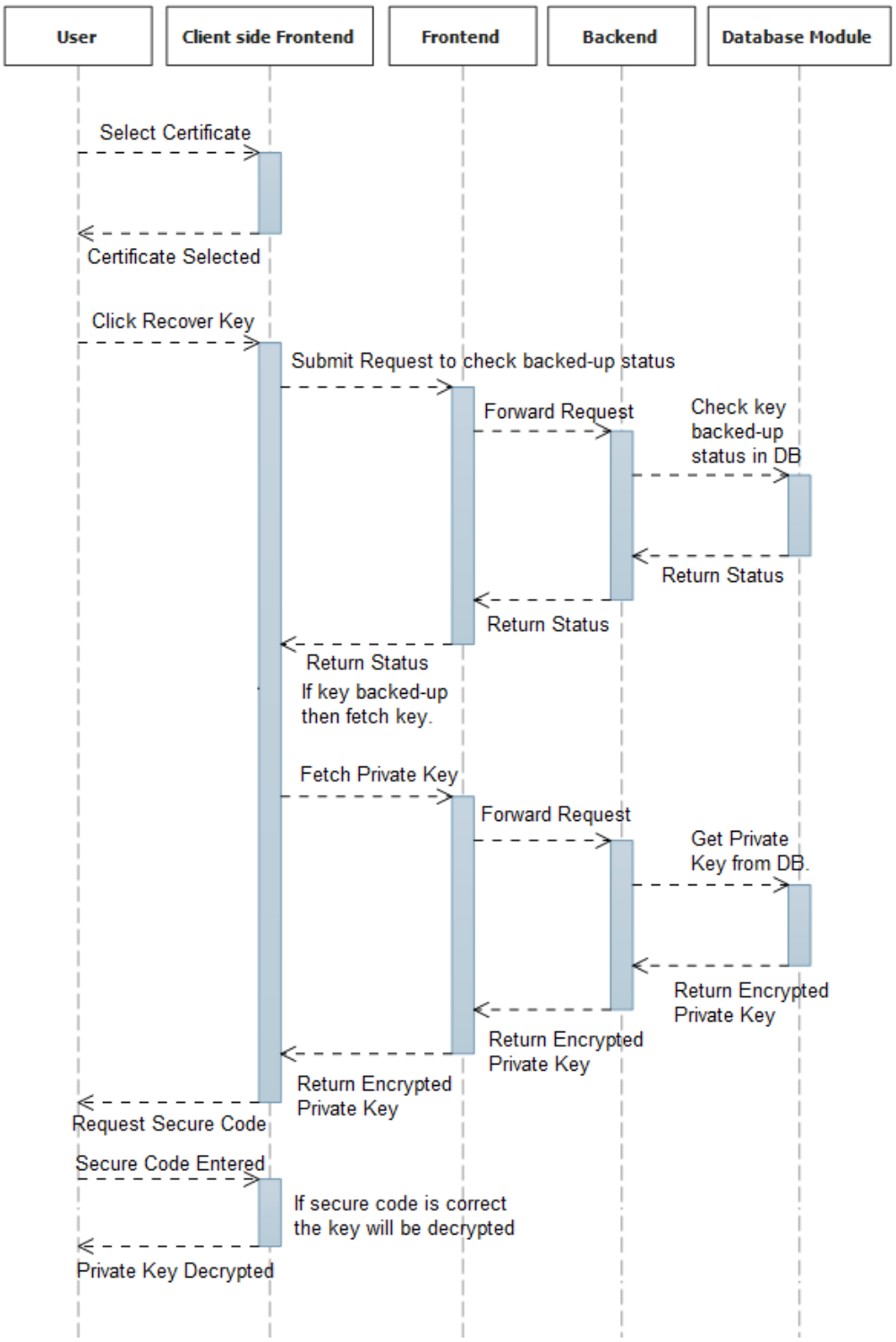

Figure 7.10: Sequence diagram depicting the process of recovery of a backed-up key from the service.

Chapter 8

Evaluation

This chapter discusses the results of the evaluation of the implemented certificate management service. The evaluation is based on how the implemented certificate management service improves upon the various steps in the life-cycle of a certificate, as compared to the systems and processes currently in use in the industry and how in some cases the implemented certificate management service overcomes some limitations of present-day systems and processes. A modified life-cycle was already discussed in the Analysis chapter. In the following sections, each step of this life-cycle is individually evaluated with respect to the certificate management service. One significant improvement that has to be noted is that the certificate management service makes the use of OpenPGP possible which many systems do not provide for.

8.1 User Registration

An end user needs to be registered with a Certifying Authority (CA) in order to send a certificate request and carry out other interactions with the CA. Usually, users do not wish to carry out this task, as the CA requires extensive user information for registration. This adds an extra usability issue, where the user may not wish to fill out yet another registration.

Sometimes, user registration even requires a personal verification and the user may have to pay a visit to the offices of a CA or an appropriate Registration Authority (RA) along with an identity proof. These offices could be located at far-off locations thus adding an accessibility issue for the end user.

As far as the implemented certificate management service is concerned, the user registration is done through the registration page. As earlier discussed, user registration should be the domain of an external authentication component, but as far as the scope of this thesis is concerned, this can be done by the users themselves. The service operator

has the option of creating new users specific to the certificate management service but that functionality is to be used in an environment where an external authentication component already exists.

When the first certificate request for a particular user is sent to the CA (depending upon approval), the user is automatically registered with the CA (implemented in this case using EJBCA). The user does not have to enter any extra credentials for this or carry out any new registration operations. Any future interactions of the certificate management service with the CA use the same user credentials to authenticate with the CA.

The limitations arising from the need for personal verification also are eliminated or are highly diminished. As long as the CA or RA trusts the implemented certificate management service, such verification and rules governing it can be left to the operators of the implemented certificate management service. Most of the verification work can be carried out, if needed, at an easily accessible location for the end user. The rest of the verification process can be automated by the implemented certificate management service with respect to the CA or the RAs.

8.2 Certificate Request

Systems, similar to the implemented service, in use today tend to allow users to request certificates by either generating a certificate signing request within a browser or uploading a certificate signing request to the system. Many a time both the methods are allowed but sometimes, only one of these methods is allowed.

The implemented certificate management service allows for both methods to be used. The end user can request a certificate by generating a certificate signing request in the browser itself and then forwarding it to the service for approval. Apart from this, the end user can also request a certificate by uploading a certificate signing request that he or she may have generated using other means.

8.3 Certificate Approval

Many a time, certificate management services and similar systems leave the task of approving a certificate on to the CA or the RAs. In these cases, Identity checks are not performed before forwarding a certificate request to a CA or a RA. If the end user sends forged data or even malicious data to the CA, the CA does not have a mechanism to verify if the data is correct.

In the implemented certificate management service, a certificate signing request passes through two levels of approval:

1. In the first level, once a certificate request has been received, the Certificate Policy Module in the back-end of the certificate management service carries out certain semantic checks on the request. It extracts the data inside the certificate signing request to verify if it matches with the known identity of the user. Ideally, this identity and related data should be provided by an external component - the Identity Management System. However, as far as the scope of this thesis is concerned, it is done using the identity information which the user provided upon registration. It also checks if the request meets requirements for the key algorithm used and the key length used. If the Certificate Policy Engine checks pass, then the request is stored in the appropriate table.
2. The second and last approval rests with the Support Staff who can either approve or decline a request.

8.4 Certificate Issue

In the implemented certificate management service, once a certificate request has been approved by the Support Staff, it is forwarded to the CA. The CA in turn, issues the certificate which is then stored in the database of the certificate management service as well as the local repository of the CA. This process follows the same methodology as processes currently in use in the industry.

8.5 Certificate Enrolment

Under this step, once the certificate is issued, it can be enrolled. The CA can post it to a repository for example, and send an identity certificate back to the user that he or she can now use. Present systems enroll certificates in this way.

As far as the implemented certificate management service is concerned, the CA (implemented using EJBCA) carries out the enrolment process itself. It also sends back the issued certificate to the certificate management service. The certificate is stored in the database and presented to the end user on his or her home page. Hence, the implemented certificate management service is in line with the current practices.

8.6 Certificate Distribution

Systems currently in use in the industry, distribute certificates by putting them in a repository. The certificates can be further distributed to a user using that repository.

The implemented certificate management service follows a similar process. It holds all issued certificates in a database table. Similarly, for all signed OpenPGP Keys, there is another table in the database. These tables can be accessed through the API of the service. Thus, certificates and OpenPGP Keys can be distributed easily to an external certificate directory or key server respectively. On an end users home page, the user is presented with his or her issued certificates and signed OpenPGP keys (and the status for each) by fetching needed data from the same tables.

8.7 Certificate Modification

Current systems, hardly provide the functionality to modify certificate data. This is because previous certificate signing requests or certificates cannot be changed without invalidating the respective signatures on them. This brings down the usability of such systems as a change in user specific data in certificates is a recurring phenomena in the real world. For example, A change of name upon marriage or a change of departments within an organization.

In the implemented certificate management service, this limitation is circumvented by following a simple process. The user can modify the data in a certificate that has been issued to him or her. First, the end user can generate a new certificate signing request with the changed data and submit it. This request is then sent to the Support Staff for approval. Once approved by the Support Staff, the old certificate is revoked and a new one is issued to the end user.

8.8 Certificate Renewal

Current systems, many a times, do not provide for any automated way of notifying an end user about the expiration of his or her certificates. An end user may not be aware and his or her certificate may lapse its validity period and expire.

When it comes to the implemented certificate management service, it takes care of notifying the users about their certificates which are nearing expiry. Upon user log-in, his or her issued certificates are presented on the home page in a tabular form. Each certificate's expiry date is also presented here. If a particular certificate is nearing expiry, the user is initially notified by a pop-up message upon log-in. The user can then chose to renew his or her certificate (if needed). This request then needs to be approved by the Support Staff before the Certificate can be renewed. In this case, the old certificate is revoked and the previously generated certificate signing request is re-submitted to the CA for issuing a new certificate. All this is done automatically by the certificate management service once a renewal request is approved by the support staff.

The API of the certificate management service also provides for API calls that can check if a certificate is nearing expiration.

8.9 Certificate Revocation

Usually, when an end user wishes to revoke his or her certificate it can be a tedious task. One reason for this is trying to understand what reasoning to provide to the CA for revocation. Also, at times, users fail to understand the implications of revocation. Another reason is finding the appropriate way to request a revocation. Many a times, the end user may need to write an E-Mail to the administrator of a CA or a RA to do this. In short, the starting point to begin the whole revocation process maybe difficult to come by for a normal user. This affects the usability and efficiency of the process of revoking certificates and consequently that of the system itself.

The implemented certificate management service tackles both these issues. Upon user log-in, his or her issued certificates are presented on the home page in a tabular form. Each certificate's current status can be seen here. If the user wishes to revoke a certificate, he or she can press the revoke button for that certificate. This makes the whole task of revoking certificates easy for the user as the user does not have to look for the appropriate method or contact-person to get his or her certificate revoked. The user is then presented with certain information explaining to him or her the consequences of this action. Here, the end user can also enter the description for the reason why he or she wishes to revoke the certificate.

The request is then sent to the support staff for approval. Upon approval, the support staff then chooses the appropriate reason from a list of revocation reasons and sends the request to the CA. This makes it easier for the end-user who may not be so informed about what the different reasons for revocation could be (as defined by RFC 5280). However, the Support Staff is knowledgeable about them and can choose the appropriate reason based on the description provided by the user. A revocation request is then sent to the CA or the RA. Once, the certificate is revoked, this same table on the user home page shows the status of the Certificate as revoked.

8.10 Certificate Expiration

If the end user does not renew his or her certificate well within the validity period then that certificate will be shown as expired in the table on the end user's home page.

8.11 Summary of the results of the evaluation

Hence, it can be easily understood, that the implemented certificate management service significantly improves in many ways over systems currently used in the industry. It provides for better usability and more automation of processes thus making the whole process of secure communication comparably easier for the end user. It has a significant advantage when it comes to technologies supported over many of its counterparts as it not onlz supports X.509 certificates but also OpenPGP certificates.

Chapter 9

Conclusion

This thesis analysed how an effective, usable and secure system can be implemented for big organisations so as to further the goal of secure communication. It discussed the design and implementation of a management service for digital certificates which uses both of the established technologies in the field - X.509 and OpenPGP. It discussed the current technologies, protocols and solutions available for certificate management. It elaborated on these technologies like S/MIME and OpenPGP and their related processes like Encryption and Digital Signatures.

The thesis also analysed the certificate life-cycle and elaborated upon it by putting forth a modified certificate life-cycle that was used in designing the certificate management service and then consequently evaluating it.

Another key finding of the thesis was the result of the analysis of similar certificate management systems that are in use in the industry today. It was found out that all investigated systems do not support OpenPGP and are limited in their use by supporting only X.509. Many usability issues were also found in these systems. While the Enterprise PKI with Windows Server was not platform independent and was specific to the Windows Server Operating System, OpenCA and Dogtag CA proved to be limited by the absence of any non-complicated way to communicate with them over a network or by the extreme difficulty it takes to set them up.

The thesis laid out the work-flows that would be needed in a certificate management service, after studying all possible use case scenarios for the certificate management service. It established the user base that a certificate management service would have and laid down functional requirements of each type of user. The functional requirements for the thesis were thus divided on the basis of the types of users - the system administrators, the end users, the support staff, the service operators, the recovery operators and the external users. The thesis found that although specific users are needed for recovery of certificate keys i.e. the recovery operators, their task requires additional processes to be formulated. These processes also have legal implications. Hence, this set of users

was left out of the purview of the thesis in the subsequent design and implementation of the certificate management service.

With the design of the certificate management service, the thesis was able to design secure processes for certificate signing request generation, key escrow and OpenPGP key signing. These processes were then implemented using state of the art technologies and cryptographic methods. One key finding here was that key escrow proved to be a difficult process to implement in an entirely secure manner. A secure implementation is impossible because the onus of providing verified and correct private key is on the client system which ideally should not be trusted. Hence, the key escrow mechanism implemented in the certificate management service, although up to the mark in comparison with industry wide standard practices, is not completely secure.

Upon evaluation of the implemented certificate management service, the thesis found that, in many cases, the implemented certificate management service significantly improves upon similar systems that are deployed in the industry today. The implemented certificate management service provides a more easily usable service for normal users which covers both the established technologies - X.509 and OpenPGP. Most systems fail to provide both these technologies while others are difficult for a user to use.

In conclusion, this thesis was able to realise an effective, usable and secure certificate management service. Researchers and readers alike would be able to get valuable ideas about the design and realization of a certificate management service through the chapters of this thesis. Organizations would be able to use the design for the certificate management service presented in the thesis to implement their own certificate management services. Organizations can also use the implemented certificate management service along with a few changes to meet their own requirements. The implemented certificate management service can also prove as the basis for a larger system comprising of a certificate directory service and OpenPGP key servers.

9.1 Future work

This section outlines future work that can further build upon the implemented certificate management service and improve it.

9.1.1 Key Pair verification on the Server-side

At the moment, the certificate management service verifies a user's public private key pair on the client side i.e. in the Frontend. When a user wishes to back-up his or her private key at a later stage, the service verifies if the key belongs to an already uploaded certificate or not but this verification process is done on the client-side in the clients browser. This will require the private key to be first transmitted to the server and in no

way the key can be encrypted before this transmission. Hence, the whole purpose of secure key back-up where the server cannot in any way view the private key data is defeated.

Adam Young and Moti Yung introduced the idea of Auto-Recoverable and Auto-Certifiable Cryptosystems in their paper [35] to address the issue of key escrow where in the keys can be prevented from being maliciously tampered with. However, this in itself is a work in progress, and little progress has been made on the topic since 1999. This idea laid the foundations for the idea of Publicly Verifiable Secret Sharing (PVSS) schemes. Any secret sharing scheme can be a PVSS Scheme if it is a verifiable secret sharing scheme and if any party involved in the scheme can verify the validity of the shares distributed by another participant in the scheme. This would make it feasible for the back-end of the certificate management service to verify if the private key being uploaded to the server is actually the key belonging to the certificate it is being uploaded for. All this would be done without the server actually being able to see the contents of the key. A PVSS scheme can be set up for the certificate management service in the future, thus preventing users from backing-up malicious or incorrect keys to the certificate management service.

9.1.2 Backup and Disaster Recovery

The service operators should have the functionality to back-up the entire database of the certificate management service and be able to recover the database from these back-ups when needed. This functionality was left out of the design and implementation of the certificate management service as it was out of the scope of this thesis. This functionality can be added as a future addition. This addition would require some technical requirements to be met like backup servers.

9.1.3 Recovery Operators

Although recovery operators and their required functionalities were discussed in the Analysis chapter, they were left out of the design and implementation part. This was primarily because recovery operators require a secure process by which they can recover keys for certificates that have been issued. This has legal implications as well. In the future, these functionalities can be added once such a secure process is formulated.

Bibliography

[1] D. Crocker, "Email history," *The living Internet*, 2006.

[2] C. Kaufman, R. Perlman, and M. Speciner, *Network security: private communication in a public world*. Prentice Hall Press, 2002.

[3] P. R. Zimmermann, *The Official PGP User's Guide*. Cambridge, MA, USA: MIT Press, 1995.

[4] A. S. Tanenbaum, *Computer Networks*, 3rd ed. Prentice Hall PTR, 1996, ch. 7, pp. 663–667.

[5] A. Whitten and J. D. Tygar, "Why johnny can't encrypt: A usability evaluation of pgp 5.0," in *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, ser. SSYM'99. Berkeley, CA, USA: USENIX Association, 1999, pp. 14–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251421.1251435

[6] S. A. Brands, *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. Cambridge, MA, USA: MIT Press, 2000.

[7] C. Slobogin, "Transaction surveillance by the government," *Miss. LJ*, vol. 75, p. 139, 2005.

[8] J. C. Sipior and B. T. Ward, "The ethical and legal quandary of email privacy," *Communications of the ACM*, vol. 38, no. 12, pp. 48–54, 1995.

[9] G. Greenwald, E. MacAskill, and L. Poitras, "Edward snowden: the whistleblower behind the nsa surveillance revelations," *The Guardian*, vol. 9, no. 6, 2013.

[10] G. Greenwald and E. MacAskill, "Nsa prism program taps in to user data of apple, google and others," *The Guardian*, vol. 7, no. 6, pp. 1–43, 2013.

[11] J. Yu, V. Cheval, and M. Ryan, "Challenges with end-to-end email encryption."

[12] S. Ruoti, J. Andersen, D. Zappala, and K. Seamons, "Why Johnny Still, Still Can't Encrypt: Evaluating the Usability of a Modern PGP Client," *ArXiv e-prints*, Oct. 2015.

[13] W. Stallings, *Cryptography And Network Security - Principles and Practice*, 1st ed. Dorling Kindersley India Pvt. Ltd., licensees of Pearson Education in South Asia, 2011, ch. 14, pp. 453–463.

[14] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280 (Proposed Standard), Internet Engineering Task Force, May 2008, updated by RFC 6818. [Online]. Available: http://www.ietf.org/rfc/rfc5280.txt

[15] P. Resnick, "Internet Message Format," RFC 2822 (Proposed Standard), Internet Engineering Task Force, Apr. 2001, obsoleted by RFC 5322, updated by RFCs 5335, 5336. [Online]. Available: http://www.ietf.org/rfc/rfc2822.txt

[16] C. Adams, S. Farrell, T. Kause, and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)," RFC 4210 (Proposed Standard), Internet Engineering Task Force, Sep. 2005, updated by RFC 6712. [Online]. Available: http://www.ietf.org/rfc/rfc4210.txt

[17] M. Myers, X. Liu, J. Schaad, and J. Weinstein, "Certificate Management Messages over CMS," RFC 2797 (Proposed Standard), Internet Engineering Task Force, Apr. 2000, obsoleted by RFC 5272. [Online]. Available: http://www.ietf.org/rfc/rfc2797.txt

[18] R. Housley, "Cryptographic Message Syntax," RFC 2630 (Proposed Standard), Internet Engineering Task Force, Jun. 1999, obsoleted by RFCs 3369, 3370. [Online]. Available: http://www.ietf.org/rfc/rfc2630.txt

[19] M. TechNet. (2006, aug) Understanding s/mime. [Online]. Available: https://technet.microsoft.com/en-us/library/aa995740%28v=exchg.65%29.aspx

[20] R. Housley, "Cryptographic Message Syntax (CMS)," RFC 3369 (Proposed Standard), Internet Engineering Task Force, Aug. 2002, obsoleted by RFC 3852. [Online]. Available: http://www.ietf.org/rfc/rfc3369.txt

[21] R. Housley, "Cryptographic Message Syntax (CMS) Algorithms," RFC 3370 (Proposed Standard), Internet Engineering Task Force, Aug. 2002, updated by RFC 5754. [Online]. Available: http://www.ietf.org/rfc/rfc3370.txt

[22] B. Ramsdell, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling," RFC 3850 (Proposed Standard), Internet Engineering Task Force, Jul. 2004, obsoleted by RFC 5750. [Online]. Available: http://www.ietf.org/rfc/rfc3850.txt

[23] B. Ramsdell, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification," RFC 3851 (Proposed Standard), Internet Engineering Task Force, Jul. 2004, obsoleted by RFC 5751. [Online]. Available: http://www.ietf.org/rfc/rfc3851.txt

[24] W. Stallings, *Cryptography And Network Security - Principles and Practice*, 1st ed. Dorling Kindersley India Pvt. Ltd., licensees of Pearson Education in South Asia, 2011, ch. 18, pp. 591–627.

[25] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, "OpenPGP Message Format," RFC 4880 (Proposed Standard), Internet Engineering Task Force, Nov. 2007, updated by RFC 5581. [Online]. Available: http://www.ietf.org/rfc/rfc4880.txt

[26] N. Associates, *PGP 6.0.2 Platform-Independent Documentation: PGP Installation Guide, an Introduction to Cryptography, Administration's Guide.* Network Associates, 1998, ch. 1. [Online]. Available: https://books.google.de/books?id=ghG4AAAACAAJ

[27] Y. Chou. (2013, oct) Enterprise pki with windows server 2012 r2 active directory certificate services (part 1 of 2). [Online]. Available: https://blogs.technet.microsoft.com/yungchou/

[28] M. TechNet. (2008, mar) Active directory certificate services overview. [Online]. Available: https://technet.microsoft.com/en-us/library/a8f53a9b-f3f6-4b13-8253-dbf183a5aa62.aspx

[29] P. S. AB. (2016) Ejbca - open source pki certificate authority - home. [Online]. Available: https://www.ejbca.org/

[30] A. I. Ghori and A. Parveen. (2006) Pki administration using ejbca and openca. [Online]. Available: http://ece.gmu.edu/coursewebpages/ECE/ECE646/F09/project/reports_2006/IL-3-report.pdf

[31] O. Group. (2016) Openca research labs - home page. [Online]. Available: https://www.openca.org/projects/openca/

[32] C. C. M. P. U. B. A. J. M. Bartosch, M. Bell, *OpenCA Guide for Versions 0.9.2+*, OpenCA Group, 2005. [Online]. Available: "http://www2.openxpki.org/docs/guide/openca-guide.pdf"

[33] Spring, "Springframework reference manual 3.1," 2011.

[34] GlobalSign and P. Ventures. (2016). [Online]. Available: https://pkijs.org/

[35] A. Young and M. Yung, *Auto-recoverable auto-certifiable cryptosystems.* Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 17–31. [Online]. Available: http://dx.doi.org/10.1007/BFb0054114