Remote Qualified Digital Signatures

Pedro Dias do Vale

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Ricardo Jorge Fernandes Chaves

Examination Committee

Chairperson: Prof. Ana Teresa Correia de Freitas
Supervisor: Prof. Ricardo Jorge Fernandes Chaves
Member of the Committee: Prof. Luís Brás Monteiro Guerra e Silva

November 2017
Acknowledgements

I would like to thank all the people that, in some way, contributed to my academic path.

First of all, to my parents for supporting me throughout all these years. I also would like to thank the remainder of my family, specially my sisters and to my brother-in-law, Nuno Coração.

I would also like to thank my dearest friends. The people from FCUL, for providing me the strongest experiences a person should have and a lot of happy moments. This includes: Marta, BdN, Desmarques and GPS. Special thanks to Maias and DaCosta, companions of jokes and random conversations.

Then, to José Nogueira and José Amaral, for being good colleagues to work as a group and good friends.

Also, to Filipe Custódio, for being in the same boat and not jumping off mid-trip.

To all the people from Contraste, now and before, for providing me awesome musical experiences.

Then to all my friends in Alenquer, the oldest in the book and companions in life-defining adventures.

Last but not the least, I would like to thank all the team at Multicert, for being good sports and supporting (also technically) the development of this work.

To each and everyone, I cherish you all, thank you.

“The two most important requirements for major success are: first, being in the right place at the right time, and second, doing something about it.” Ray Kroc

Lisboa, November 2017

Pedro Dias do Vale
For my family,
Resumo

No mundo digital em que vivemos hoje em dia, serviços de *e-Commerce* and *e-Government* proporcionam aos utilizadores a possibilidade de, com conveniência, fazer as suas actividades online, evitando a inconveniência de serviços físicos. Contudo, com estas oportunidades tecnológicas, advêm questões em relação à Segurança da Informação. Informação sensível tem de ser transmitida, normalmente por redes inseguras, visto que os serviços são remotos para o utilizador.

Assinaturas Digitais Qualificadas, definidas na Directiva 1999/93/EC[1], permitem a Autenticidade da informação que é trocada entre utilizadores e serviços remotos. O objectivo deste tipo de assinaturas é torná-las equivalentes a assinaturas manuscritas, fazendo com que tenham um valor legal forte. Para criar estas assinaturas qualificadas, um Dispositivo de Criação de Assinaturas Seguro (SSCD) tem de ser usado. Além disso, o utilizador tem de conseguir controlar o momento da criação das assinaturas (para a informação que este escolher).

Esta tese propõe uma solução para a criação de Assinaturas Digitais Qualificadas Remotas, sendo o sistema resultante designado por RSIGN. A solução proposta inclui o uso de um *Hardware Security Module* (HSM), assumindo o papel do SSCD requerido e onde as assinaturas são criadas. Os utilizadores accedem ao serviço através do seu dispositivo móvel, sem a necessidade de *hardware* extra. A informação que permite a criação da assinatura é controlada de forma segura e apenas pode ser utilizada para criar assinaturas com autorização explícita do utilizador. De modo a permitir ao a escalabilidade do sistema relativamente ao número de utilizadores que este suporta, as chaves de assinatura são guardadas fora do HSM, protegidas de tal forma que apenas podem ser utilizadas pelo HSM para criar assinaturas digitais, com o código de autorização do utilizador. A solução inclui ainda mecanismos para proteger a informação trocada entre os utilizadores e componentes confiáveis do sistema, prevenindo assim ataques à comunicação a ser transmitida.
Abstract

In the digital world that we live in, e-Commerce and e-Government services pose a convenient way for users to carry out activities online, thus avoiding the nuisance of traditional physical services. However, these opportunities carry new technological challenges regarding information security. Sensitive data has to be passed around, usually through insecure networks, because these services are remote to the user.

Qualified Digital Signatures, defined in Directive 1999/93/EC[1], provide Authenticity to the data being exchanged between users and remote services. The goal of this type of signatures is to make them equivalent to hand written signature, granting them a strong legal value. To compute these qualified signatures, a Secure Signature Creation Device (SSCD) must be used. Also, the user must control when signatures, for the data it chooses to, are created.

This thesis proposes RSIGN, a solution that provides Remote Qualified Digital Signatures. It includes a Hardware Security Module (HSM), as the required SSCD, where signatures are created. The user accesses the service using its mobile device, no extra hardware is necessary. The users’ signature data is securely managed and can only be used to create signatures with explicit authorisation of the user. In order to allow the system to scale in terms of users it supports, signing keys are stored outside the HSM, protected in such a way that only the HSM can use them to create signatures, using the users’ authorisation code. The solution also includes a secure way for the users’ and trusted components in the system to exchange data, preventing attacks to the communication.
Palavras Chave

Assinaturas Qualificadas
Assinaturas Remotas
eIDAS
Dispositivos Móveis
Segurança da Informação

Keywords

Qualified Signatures
Remote Signatures
eIDAS
Mobile Devices
Information Security
# Contents

1 Introduction ........................................... 1
   1.1 Motivation ........................................ 2
   1.2 Contributions ..................................... 3
   1.3 Requirements ...................................... 4
   1.4 Document Structure ................................ 4

2 Background ............................................ 5
   2.1 Cryptographic mechanisms ......................... 6
   2.2 Cryptographic Protocols ............................ 8
   2.3 TLS .................................................. 10
   2.4 HSM .................................................. 11
   2.5 PKCS#11 .......................................... 12
   2.6 Summary ............................................ 14

3 Related Work .......................................... 15
   3.1 Qualified Digital Signatures and Certificates .... 15
   3.2 Overview ............................................ 16
   3.3 Existing solutions .................................. 17
      3.3.1 Scheme by Orthacker et al. .................. 18
      3.3.2 Scheme by Zefferer et al. ................... 19
      3.3.3 Scheme by Rossnagel ......................... 20
### 3.3.4 Scheme by Rath et al.

Page 21

### 3.3.5 Comparative Analysis

Page 22

### 3.4 Summary

Page 24

---

### 4 RSIGN

#### 4.1 Trust Model

Page 26

#### 4.2 Architecture Overview

Page 28

#### 4.3 Export Signing keys

Page 30

#### 4.4 Using signing keys

Page 32

#### 4.5 Secure Channels

Page 33

- **4.5.1** Secure channel establishment

Page 35

- **4.5.2** Secure channel usage

Page 37

- **4.5.3** Alternative approach

Page 38

#### 4.6 RSIGN’s functionality

Page 38

- **4.6.1** Signup

Page 39

- **4.6.2** Login

Page 40

- **4.6.3** Registration

Page 40

- **4.6.4** Signing

Page 41

- **4.6.5** Change PIN

Page 42

- **4.6.6** Log off

Page 43

#### 4.7 Summary

Page 43

---

### 5 RSIGN’s Implementation

#### 5.1 Prototype Implementation

Page 46

#### 5.2 RSIGN’s Technology

Page 47

#### 5.3 Communication Protocol

Page 48
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4</td>
<td>Storing signing keys</td>
<td>49</td>
</tr>
<tr>
<td>5.5</td>
<td>Interacting with the HSM</td>
<td>50</td>
</tr>
<tr>
<td>5.6</td>
<td>Client application: myRSIGN</td>
<td>53</td>
</tr>
<tr>
<td>5.7</td>
<td>Summary</td>
<td>55</td>
</tr>
<tr>
<td>6</td>
<td>Evaluation</td>
<td>59</td>
</tr>
<tr>
<td>6.1</td>
<td>Comparing RSIGN with Existing Solutions</td>
<td>59</td>
</tr>
<tr>
<td>6.2</td>
<td>Security Evaluation</td>
<td>61</td>
</tr>
<tr>
<td>6.3</td>
<td>Performance Evaluation</td>
<td>65</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Overall Performance</td>
<td>66</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Signature Creation</td>
<td>67</td>
</tr>
<tr>
<td>6.4</td>
<td>Final Considerations</td>
<td>68</td>
</tr>
<tr>
<td>7</td>
<td>Conclusions</td>
<td>71</td>
</tr>
<tr>
<td>7.1</td>
<td>Conclusions</td>
<td>72</td>
</tr>
<tr>
<td>7.2</td>
<td>Future Work</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>78</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Digital Signature creation by entity $A$, for message $M$, and verification process by entity $B$. ........................................ 7

2.2 Diffie-Hellman Key Exchange Protocol ........................................ 9

3.1 Architecture of the solution proposed by Zefferer et al.[17] ................. 19

4.1 RSIGN’s Architecture .............................................................. 28

4.2 Components included in the Client Portal and in the MTC Backend ....... 29

4.3 Communication channels between the components in the RSIGN’s architecture ....... 34

4.4 Establishment of the secure channel between the user and RSIGN Backend. ... 36

4.5 Part of Multicert’s web page with remote qualified certificate available. ....... 39

4.6 Sequence digram presenting user registration in RSIGN ......................... 40

4.7 Sequence digram presenting the signing operation ............................... 41

5.1 Implementation architecture ....................................................... 46

5.2 Java objects being exchanged between RSIGN and users’ endpoints. ....... 49

5.3 Data model for the database that stores RSIGN data. ........................... 49

5.4 Different layers involved in the communication with RSIGN’s HSM ........... 51

5.5 myRSIGN Signup ................................................................. 53

5.6 myRSIGN Main menu ............................................................ 54

5.7 myRSIGN Successful Registration .............................................. 54

5.8 myRSIGN Signing operation ..................................................... 55
5.9 myRSIGN Signing operation result .................................................. 55
5.10 myRSIGN Login error ................................................................. 56

6.1 Latency for each RSIGN operation, measured in seconds (standard deviation included). ................................................................. 66
6.2 Latency of the signing request, depending on the number of files to be signed. . 67
List of Tables

3.1 Comparative table of Signatures Creation solutions . . . . . . . . . . . . . . . . . . 22

5.1 PKCS#11 attributes[33] for keys of RSIGN . . . . . . . . . . . . . . . . . . . . . . . 52
## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CA</td>
<td>Certification Authority</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DHKE</td>
<td>Diffie-Hellman Key Exchange</td>
</tr>
<tr>
<td>DH</td>
<td>Diffie-Hellman</td>
</tr>
<tr>
<td>EU</td>
<td>European Union</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HMAC</td>
<td>Hashed Message Authentication Code</td>
</tr>
<tr>
<td>HSM</td>
<td>Hardware Security Module</td>
</tr>
<tr>
<td>HTTPS</td>
<td>HTTP over TLS</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IV</td>
<td>Initialization Vector</td>
</tr>
<tr>
<td>JCA</td>
<td>Java Cryptography Architecture</td>
</tr>
<tr>
<td>MAC</td>
<td>Message Authentication Code</td>
</tr>
<tr>
<td>MOA-SS</td>
<td>Module for Online Applications - Server Signature</td>
</tr>
<tr>
<td>MTC</td>
<td>Multicert</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>OTP</td>
<td>One Time Password</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PFS</td>
<td>Perfect Forward Secrecy</td>
</tr>
<tr>
<td>PIN</td>
<td>Personal Identification Number</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>RSIGN</td>
<td>Remote SIGNature</td>
</tr>
<tr>
<td>SIM</td>
<td>Subscriber Identity Module</td>
</tr>
<tr>
<td>SMS</td>
<td>Short Message Service</td>
</tr>
<tr>
<td>SSCD</td>
<td>Secure Signature Creation Device</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Socket Layer</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URL</td>
<td>Universal Resource Locator</td>
</tr>
<tr>
<td>XML</td>
<td>Extensive Markup Language</td>
</tr>
</tbody>
</table>
Nowadays many services are made available to users digitally. The way users can interact with these services, through their mobile devices connected to the internet, provides a convenient way for them to carry out their daily online activities. To accomplish this, the user must provide information so that services, e.g. e-Commerce and e-Government, can process it to perform the desired operations. This kind of information is, many times, sensitive information that should not be disclosed to unwanted entities. However, many cyber attacks are made possible everyday due to the lack of strong security mechanisms that the remote services should provide.

One of the requirements for services, is the Authenticity of the information being sent, i.e. the means to prove that the information was sent by a given authorised user and that it was not tampered with during the exchange. Digital Signatures can fulfil this security requirement.

Digital Signatures can be compared to hand written signatures, in the sense that they identify the signatory and that he/she had knowledge of the content is being signed. Furthermore, the signature is unique and, ideally, only its owner can present it correctly. For example, if it was not necessary to sign cheques, through hand-written signatures, an intermediary could simply request a given amount to be taken from a random account to its own and the unauthorised operation would be validated by the bank. By signing a cheque in the process of buying a product at a store, the signatory agrees on a particular value to be taken from its account and put into the store’s account. A digital signature, sent along with a message also guarantees that if the message was changed in some way during the exchange, i.e. tampered with, it does not go unnoticed during the signature verification process and that the user cannot deny having signed the document (since only he/she can present it). In the example above, an attack to the authenticity of the cheque would be changing the amount of money being taken from the account by someone other than the cheque owner.
1.1 Motivation

Digital signatures are created on behalf of the users, since these users cannot physically create them. For that reason, digital signatures must be created in a reliable manner. Considering this assumption does not always hold for most services, they cannot trust sensitive operations, such as purchase, to be performed over the internet.

Directive 1999/93/EC[1] of the European Parliament and the European Council established legal requirements for Qualified Digital Signatures. Qualified Digital Signatures, or Qualified Signatures for short, are a very reliable type of digital signatures, because of the way they are created. Even though digital signatures can be compared to hand-written signatures, only qualified signatures are actually equivalent within the European Union (EU). This means that they can authenticate information being sent to services within the EU.

One of the requirements for obtaining qualified signatures is that they must be created inside a Secure Signature Creation Device (SSCD), a hardware device that performs the necessary cryptographic operations while storing sensitive data. This requirement poses a usability limitation for users. To interact with services that require the authenticity of the information through qualified signatures, they must acquire a SSCD. As one can imagine, this requirement limits the creation of the qualified signatures in mobile platforms, such as laptops and smartphones, the first choice of platforms for most users across the world nowadays. This was not a concern in 1999, when the Directive was created. For that reason, the requirements prove to be somewhat limited and requires that users have to carry a SSCD in order to create qualified signatures using their mobile devices.

Later, Regulation No. 910/2014[2], commonly known as eIDAS (IDentification and Authentication Services), replaced the previous Directive and adapted the requirements to current mobility trends. This Regulation allows remote signature creation, as long as only users reliably authorise signature creation in their name. What this means is that it must not be possible for the service to create a qualified signature without prior authorisation of the user, nor can any other entity gain authorisation privileges.

Since 2014, there has not been many solutions that allow the user to obtain signed data in a way that he does not need to carry a SSCD, which means that there is the opportunity for new solutions to be proposed, making them commercially attractive. Usually, remote qualified
signature solutions include an Hardware Security Module (HSM) as the SSCD. An HSM is a cryptographic hardware device, which is trustworthy and resistant to physical attacks, allows for secure cryptographic operations (such as signature creation). Usually these devices have limited resources, such as storage, meaning that sensitive data necessary for the creation of signatures is usually stored externally. This requires that the data is protected in a way that only the HSM can use it to create qualified signatures.

1.2 Contributions

The proposed service, Remote SIGNature (RSIGN), is a remote HSM-based solution capable of producing qualified signatures whilst the user is remotely connected to the service, through its mobile device. For the user who is interacting with the service, it is intended to be done in a transparent manner, i.e. the HSM appears to be a local SSCD. In this solution, the user must input its authorisation code, i.e. a PIN, to authorise signature creation.

One of the security advantage of this solution is that this authorisation code is never stored in the service and is only known to the user. This is accomplished by protecting the users’ signature data using both the HSM’s secret data and the mentioned authorisation code. Only this device can have access to the data necessary to create signatures, using the users’ authorisation code, and sign whichever data that sent by them.

The exchange of the authorisation code and the sensitive information between the user and the remote HSM in RSIGN is an important issue. This exchange is even more sensitive since intermediate components, for which the information has to go through, are not fully trusted. A secure channel between the users’ endpoint and the components that directly support the qualified signatures creation, such as the HSM, must be established before any sensitive data is exchanged.

RSIGN is a product developed for Multicert, a company that provides certification services and security solutions. For this reason, the RSIGN’s design must take into account the ease of integration in the existing infrastructure. Furthermore, it must be scalable to support many users who potentially can use the remote qualified signature service.

This solution overcomes some disadvantages of the existing solutions, mainly due to its hardware and third-party independence, i.e. the user must only have its mobile device and only
access to the Multicert service to successfully obtain qualified signatures.

1.3 Requirements

The following states the requirements that the solution must satisfy:

1. The system must provide the ability to sign data remotely in a secure and reliable manner;
2. The users must be able to interact with the remote service using their mobile device;
3. The communication channels must follow the properties of Data Authenticity, Confidentiality and Freshness;
4. A remote HSM device must be the only component capable of performing the qualified cryptographic operations, namely creating the signatures;
5. The provided remote signature solution must be compliant with eIDAS[2].

To further clarify the terms mentioned in Requirement 3, ”Authenticity is the property of being genuine and being able to be verified and trusted (...)”[3] which considers both data and entity authenticity. Confidentiality is related with the data being undisclosed to unauthorised entities. Data Authenticity demands that data can only be modified in an authorised manner. Finally, Freshness means that the data being exchanged is recent.

1.4 Document Structure

This document is organised in the following way: Chapter 2 states some concepts that are necessary to have in mind throughout the rest of this thesis. Chapter 3 presents the context on the subject of Qualified Signature along with existing solutions. Section 4 presents the solution stating its core internal features and the functionality it offers to client applications. Chapter 5 presents implementation details of the proposed solution and the Client application that was developed to allow user interaction with the remote signature creation service. Section 6 presents an Evaluation of the implemented solution mainly in terms of security. Section 7 finalises with a summary of what was accomplished and comes next, given the results obtained.
After a high-level description of the problem and solution, this Section will detail related background, necessary to understand the next chapters.

This chapter begins with basic cryptographic mechanisms which are the elementary concepts in computer security (Section 2.1), namely Symmetric and Asymmetric Encryption, Hashing, Key Wrapping, Key Derivation, Digital Signatures and Digital Certificates.

In this chapter two very important protocols are detailed: The Diffie-Hellman Key Exchange Protocol and RSA (Section 2.2). These protocols allow data to be exchanged between remote entities in a reliable manner.

More complex cryptographic protocols rely on the previous mechanisms for secure channel establishment and usage. One example of this protocol is the Transport Layer Security (TLS), that allows for entities to exchange data, protected from common attacks that may happen in between the message’s exchange. This is further explained in Section 2.3.

As mentioned in the Introduction, HSMs are cryptographic devices that may be used to securely sign and store sensitive, i.e. they can be considered SSCDs. Some of their features are detailed in Section 2.4.

Regarding the interaction between systems and HSMs, manufacturers usually offer proprietary Application Programming Interfaces (APIs). Due to the low flexibility of the these interfaces, the PKCS#11 standard aims at overcoming this problem by offering a common interface that can be used by systems to embed different HSMs. PKCS#11 defines key templates containing the object’s attributes, mechanisms, and functions to interface with the functionality of such devices. Section 2.5 presents an overview of PKCS#11.
CHAPTER 2. BACKGROUND

2.1 Cryptographic mechanisms

Symmetric encryption is based on the knowledge of a shared secret key between two entities\[3\]. An entity that wishes to send an encrypted message to another uses a secret key to do it. On receiving the message, the other entity decrypts it using an equal secret key. The Confidentiality of the message, i.e. its privacy, is therefore guaranteed.

To mitigate the problem of key distribution and key management, Asymmetric encryption was proposed. Instead of using an equal keys to encrypt and decrypt data, entities that wish to send encrypted data to a certain entity must have its public key to encrypt messages. The receiving entity then uses its private key to decrypt the message. The private key is kept safe and is only known to its owner while the public key can be given to other entities.

When keys are used to encrypt other keys, the mechanism is called Key wrapping\[4\][5]. This mechanism is usually used to exchange keys in a reliable manner.

Both symmetric and asymmetric encryption allow for data to be encrypted, ensuring its Confidentiality\[3\]. However, sometimes the Integrity, also known as Data Authenticity, is needed. This property allows for unauthorised changes in messages to be detected. Usually this is accomplished by sending a smaller data object along with the message that is used by the receiver to verify the message’s Authenticity.

Data Authenticity is usually accomplished by Hashing functions. Hashing functions take as input, at least, the message to be sent and generate a hash value that is smaller when compared to the message itself. This compression is very important to reduce the overhead of applying this mechanism.

Message Authentication Codes (MACs) includes both symmetric encryption and hashing. After the hash of a given message is calculated, its value is encrypted with the secret key so that only the intended receiver can know it. If the message is also encrypted, both the message’s Confidentiality and Authenticity are guaranteed.

If the hash value of a given message is afterwards encrypted with a given user’s private key, the result is a Digital Signature\[3\]. Unlike the MAC, digital signatures also prevent the entity that created a signature from denying having signed it because only this entity had the private key necessary to do it. This property is called Non-Repudiation and is also valid for handwritten
signatures. To verify a signature of a given message, $M$, the receiver uses the sender’s public key, $K_{a_{pub}}$, to decrypt the hash and then it can verify the message’s Authenticity by comparing the hash value with a newly calculated one, as illustrated in Figure 2.1. If the message is also encrypted, its Confidentiality, its Data Authenticity, and its Non-Repudiation is guaranteed.

![Figure 2.1: Digital Signature creation by entity A, for message M, and verification process by entity B.](image)

The public keys that allow encryption and signatures verification, through asymmetric encryption, should also be authentic, otherwise an attacker, for instance $C$, could announce to $B$ its public key stating that it is from $A$ and this way, $B$ would exchange data that is only apparently protected from unauthorised entities. Digital certificates are documents that overcome this by providing authenticity to the public key (which is included in the digital certificate). Certificates are issued by third-parties, called Certification Authorities (CAs), which are trusted by the general community, and are signed using the CA’s private key. The entities in the community verify the certificate validity using the CA’s public key[3].

Another well-known mechanism is Key Derivation. Key Derivation Functions are used for obtaining one or more cryptographically secret keys based on the value of a key or an user’s password[6][7]. Key derivation can be used to derive the key that allows for the decryption of data, without having to store the encryption key itself.
2.2 Cryptographic Protocols

The Diffie-Hellman algorithm, commonly known as Diffie-Hellman due to the authors surname, is a key exchange technique that enables two users to establish a common key that they are going to use to encrypt the communication with[3]. Diffie-Hellman relies on the mathematical properties of the function mod because it is hard to, from the final result, revert to the original values used as parameters on the function.

Two of these original values are previously known to the entities which intend to establish a secret key (they are public or are sent in the first interaction between two parties): a prime number \( q \) and an integer \( \alpha \) which are mathematically related to \( q \). The Diffie-Hellman algorithm, used to establish a secret key, \( K \), between two parties, \( A \) and \( B \), is as follows:

- \( A \) selects a random integer lesser that \( q \), \( X_a \) and computes \( Y_a = \alpha^{X_a} \mod q \)
- \( B \) does the same to generate \( Y_b \).
- \( X_a \) and \( X_b \) are \( A \) and \( B \)'s private values and are never exchanged.
- \( A \) sends \( Y_a \) to \( B \) and \( B \) sends \( Y_b \) to \( A \).
- \( A \) computes \( Y_b^{X_a} \mod q \) and \( B \) computes \( Y_a^{X_b} \mod q \) which results in the same value, the key.

Since \( X_a \) and \( X_b \) are kept private an attacker would only have access to the values that were exchanged and the values that are made publicly available prior to the exchange: \( \alpha, q, Y_a, Y_b \) it would be necessary to reverse \( Y_b \) to get to \( X_b \) which again, and considering the use of large prime numbers, it is infeasible. Figure 2.2 illustrates the protocol.

The problem with the Diffie-Hellman scheme is that it lacks Authenticity, which means that an attacker may pretend he/she is one of the authorised parties and establish a common secret with \( A \), for instance. It may also do the same with \( B \) and be able to see in plain text all communication between \( A \) and \( B \) (this is called the man-in-the-middle-attack). In order to overcome this, authentication mechanism must be added. For example, digital signatures can be used to authenticate the data being exchanged (requires a third-party interaction), like in and Kerberos[8].
2.2. CRYPTOGRAPHIC PROTOCOLS

Diffie-Hellman may also follow Perfect Forward Secrecy (PFS). PFS is when the attacker is unable to get access to previous private values necessary to establish previous sessions even if he/she has access to the public values. In Diffie-Hellman, this can be done under the assumption that the private values $X_a$ and $X_b$ are deleted after session establishment. This version of Diffie-Hellman is called Ephemeral Diffie-Hellman.

Diffie-Hellman is a protocol that allows for agreeing on shared secret between two entities, but it is not the only way to distribute keys among entities. RSA\[9\] (name given due to the creators Ron Rivest, Adi Shamir, and Len Adleman) is the most widely accepted Public-key encryption approach\[3\]. It is intended for a party to generate a key pair, that can then publish its public key to receive encrypted communication or provide digital signatures. RSA also allows for distributing a shared secret, protecting it using the asymmetric key pair.

The algorithm relies on the difficulty of factoring the product of two large prime numbers. Reversing the algorithm to obtain the plain text of an encrypted message would be infeasible without access to the private key.

RSA uses two private values $p$ and $q$, both distinct prime numbers which are kept secret. The strength of the algorithm is on the difficulty of factoring the product of both these numbers, which is known as the "factoring problem" The algorithm works in the following way:

- Select $p$ and $q$;
- Calculate $n = p \times q$;
CHAPTER 2. BACKGROUND

- Calculate $\phi(n) = (p - 1)(q - 1)$;
- Select $e$ such that it is a relative prime\(^1\) and $e < \phi(n)$;
- Determine $d$ such that $(d * e) \mod \phi(n) = 1$ where $d < \phi(n)$;

The resulting public key is $K_{pub} = e, n$ and the private key is $K_{priv} = d, n$. The sender encrypting $M$ computes $M^e \mod n$ which results in cipher text $C$. The receiver in turn, does $C^d \mod n$ which results in the original $M$.

This solution allows the sender to authenticate and encrypt the message it is sending. An attacker that wishes to read a message would have to reverse the operations by discovering $d$ and $n$ which would involve reversing the operation which determines $d$ and get access to $p$ and $q$ which is not easy, computationally speaking.

Regarding the length of the keys that are generated, nowadays it is recommended that the use of 2048-bit keys. This is due to the possibility of brute force attacks (attacks that try every possibility of a key until it is discovered).

2.3 TLS

TLS is a protocol that provides privacy, data authenticity and freshness to message exchanged between two parties. This protocol allows for parties to choose different algorithms for key distribution, encryption and hashing when the channel is established.

TLS uses Transmission Control Protocol (TCP) and provides a reliable end-to-end service. It consists of two layers of protocols, the Record Protocol provides compression encryption and integrity properties to upper layers. The other protocols in the upper layer are the Handshake Protocol which establishes the secure connection, the Change Cipher Spec Protocol and the Alert Protocol which are used in the management of TLS.

The record protocol is the presentation layer protocol that ensures security to services in the application layer. TLS provides confidentiality (handshake protocol results in a shared secret key which is used to encrypt TLS payloads) and message integrity (also in handshake protocol a shared secret key is generated that is then used to compute a MAC). This protocol takes a

\(^1\) a and $b$ are relative primes if the only positive number that divides them is 1.
message to be send, fragments the data into blocks, optionally compresses it, applies a MAC, encrypts (using symmetric encryption), adds a header and then sends the result in a TCP segment.

The Handshake Protocol allows the server and client to authenticate each other. It is accomplished by a series of messages (in a fixed format) exchanged between the client and the server. This protocol consists of four phases. Phase one is when both parties, client and server, state their capabilities regarding TLS which includes the protocol version and compression method. Phase two is when the server sends its certificate to authenticate itself towards the client, optionally the server may request the clients’ certificate. In Phase three, the client sends its certificate if requested (for mutual authentication), and performs the key exchange, depending on the algorithm that both parties agreed, to achieve that goal (it may be the Diffie-Hellman, for instance). In the last phase, Phase four, the change cipher suite is sent by the client with encryption already being used.

Regarding uses of TLS, HTTP over TLS (HTTPS) is used by almost all websites. HTTPS, first documented in [10], is a combination of Hypertext Transfer Protocol (HTTP) and TLS to implement secure communication between a Web browser and a Web server[3]. It is widely used and it depends on the Web server supporting TLS.

2.4 HSM

"An HSM [Hardware Security Module] often is a small computer of its own, installed as a peripheral on a larger host"[11]. Although, for reasons of size and performance, is a less capable computer so it is important to consider that its Operating System (OS) may be very limited, Random Access Memory (RAM) may have little space and sometimes doesn’t even have a hard disk. Despite the possible architectures, HSMs usually include a Central Processing Unit (CPU), a RAM, a Hardware-based random generator, an Exponentiation Engine (for public key cryptography), a Symmetric Engine (for symmetric key cryptography) and a non-volatile memory.

When considering HSMs, one has to wonder why is it necessary to install a specialised hardware module in a system instead of using a single computing system to carry out the tasks. The main motivation for the use of an HSM is security. Inside the HSM critical information
assets are stored, such as private keys and digital certificates, and cryptographic operations are computed. By having these assets and operations in an isolated environment where the integrity of the cryptographic algorithms is guaranteed, the HSM-based system may trust that they are secure. Also, access to the HSM functionality is usually protected by strong authentication mechanisms ensuring that it may be limited to a defined set of entities.

In some cases it may not be possible to do cryptographic operations on a standard CPU. Many times, cryptography requires randomness (e.g. for generation of symmetric keys or nonces). A standard machine is a deterministic process, so it can create random values, for example, by harvesting inputs (such as disk events) and mix them somehow but the resulting value is not really a random value, since it can be predicted. An HSM may contain a specialised hardware-based random generators to help with this issue. These generator avoids attackers from guessing random data.

An HSM is usually said to be tamper proof however this can only be achieved to a certain point. Instead, it can be said that there are implemented techniques to ensure the following qualities:

- Tamper Resistance: it’s physically hard to break in the device;
- Tamper Evidence: is related with creating evidence of tampering with the HSM.
- Tamper Detection: it relates with the detection of tampering when it is happening.
- Tamper Response: is about taking defensive action when an attack occurs. Techniques for this issue include zeroing all sensitive data in the HSM, for example.

HSMs can also be compared to smart cards in the sense both these devices can store keys. However, smart cards are usually used to hold information necessary for validating an operation rather than performing demanding cryptographic tasks, like the HSM does.

2.5 PKCS#11

PKCS#11 (Public Key Cryptography Standards) is standard that defines an API, called "Cryptoki"[12] (which is read "Crypto Key"), to devices that hold cryptographic information and perform cryptographic functions. In this standard, these cryptographic devices are commonly
called "tokens". PKCS#11's goal is to provide a logical view of the device to applications, abstracting it from the actual hardware. This abstraction results in interoperable solutions, i.e. hardware independent.

In PKCS#11, the token is made available to applications through slots. Different applications may share the same token while having the separation of privileges limited to a slot (for each a Personal Identification Number (PIN) authentication may be defined).

One important concept in PKCS#11 is the session. A session is the period in which an application is connected to the device, through Cryptoki. To establish a session, an application must be authenticated towards the token. This authentication is usually based on the knowledge of a PIN. Cryptoki is object-based and considers two types of objects: token objects and session objects. Token objects are the objects that are stored in the token, they are persistent and usually sensitive objects. Securely storing these objects is the main goal of having a token in the first place. Session objects are temporary and its lifetime depends on the session duration, after a session is closed, the object and all objects associated to it are destroyed.

There are different kinds of objects defined in PKCS#11: Data, Keys (which can be specified to Private, Public or Secret keys) and Certificates. These objects are managed by the application using templates. Templates contain the attributes of each object. These attributes may be generic, i.e. present in different templates, or can be specific for an object, e.g. RSA public key size.

Cryptoki consists of many functions regarding the interaction between tokens and applications. These functions can be divided and include (among others) the following categories: General Purpose Functions (such as C_Initialize), Session Management functions (such as C_Login and C_OpenSession), Object Management Functions (such as C_SetAttributeValue), Encryption and Decryption functions, Signing functions, Key Management functions (such as C_GenerateKeyPair or C_WrapKey) and finally Random Number Generation functions. It is through these functions that an application may request key operations using a given algorithm as parameter (in PKCS#11 these are called mechanisms) and a key template. For example, for generating a secret key, the function would look something like this: C_generateKey(<key template>,<mechanism>). In the standard attributes start with CKA_ and mechanisms start with CKM_ (e.g. CKA_EXTRACTABLE, CKM_RSA_PKCS)

PKCS#11 also defines error types for when something goes wrong in a given function. These
errors state some information regarding what might have gone wrong, e.g. CKR_TEMPLATE_INVALID is returned when the same attribute is repeated for a given template.

2.6 Summary

This chapter provided a brief context on the concepts that are used throughout this thesis.

Cryptographic mechanisms and the security properties they guarantee were mentioned. These mechanisms are related to digital signatures in general, which is a constant throughout this thesis.

Diffie-Hellman is a key exchange protocol that allows for two entities to agree on a shared secret key, without explicitly exchanging it, that can be used to protect the privacy of thereafter communication. This mechanism lacks message authenticity. For that reason, extra mechanisms are usually added to it. RSA is the most used asymmetric encryption scheme, it relies on mathematical functions to encrypt or sign data, using a public and a private key.

TLS establishes a secure channel that provides Confidentiality, Authenticity and Freshness to the data being sent through it. It authenticates one or both of the entities using certificates.

A HSM is a cryptographic device which is usually embedded in broader systems to isolate the cryptographic operations that need to be made. Depending on the device, it may be considered a SSCD.

To interact with the HSM, a PKCS#11 interface is usually used. It abstracts the device from applications allowing for different devices to be changed. This interface allows for an application to be authenticated towards the cryptographic device, in which case it requires a PIN.
This document presents a particular solution for the problem of remote qualified digital signatures. However, this problem has already been addressed before. This section presents some context on the subject at hand (Section 3.1 and Section 3.2) and existing solutions (including a comparative analysis between them) (Section 3.3).

The following separates the solutions into two categories: server based and mobile device based signatures. There are some doubts about signature creation at the server side, when considering qualified signatures. The questions are justified with the requirements of the older Directive[1].

The four existing solutions herein presented are mostly based on smart cards where signatures may be created in there or the smart card is only used as an authentication token to authorise signature creation at the server side.

### 3.1 Qualified Digital Signatures and Certificates

As mentioned above, one of the requirements for the presented solution in this document (Requirement 5) is the use of qualified digital signatures. To explain what are qualified signatures, it is important to understand what are advanced signatures since Qualified Signatures are also Advanced Signatures. As defined in the EU Signature Directive 910/2014[2]:

“An advanced electronic signature shall meet the following requirements:

(a) it is uniquely linked to the signatory;

(b) it is capable of identifying the signatory;

(c) it is created using electronic signature creation data that the signatory can, with a high level of confidence, use under his sole control; and
Qualified signatures are advanced signatures that have more requirements. They have to be created by a Secure Signature Creation Device (SSCD) and have to be explicitly authorised by the owner of the signing key (unlike advanced signatures that can be used by automated services). In this case, not even the user has access to the private key necessary to create the qualified signature, since they are secured by the SSCD. The user can only make requests for his/her private key to be used for signing operations.

### 3.2 Overview

Regarding mobile signing infrastructures, Fritsch et al.[13] divide solutions into two categories: Server based and Mobile device based electronic signatures. When discussing Server based solutions, the certificate which holds the public key necessary to verify the signature may be issued in the name of the user or in the name of the service provider to sign documents in the name of the user. In the latter, the authors argue that the user must give away its private key which goes against the requirements for advanced signatures because data is being signed in the name of the service provider and not in the name of the user, losing its potential for qualified status (and the non repudiation property by the user, for that matter). The same argument is stated by Rossnagel[14] and by Ruiz-Martínez et al.[15].

The issue regarding the dubious status of qualified signatures when the signature is created at the server is justified with Art 2.2(c) of the EU Directive 1999/93/EC, which states that "it is created using means that the signatory can maintain under his sole control;"[1]. However, with the EU Regulation 910/2014, also known as eIDAS, released as a replacement to the previous Directive, this issue is clarified. eIDAS states that:

"(51) It should be possible for the signatory to entrust qualified electronic signature creation devices to the care of a third party, provided that appropriate mechanisms and procedures are implemented to ensure that the signatory has sole control over the use of his electronic signature creation data, and the qualified electronic signature requirements are met by the use of the device."
3.3 Existing solutions

In the following some existing solutions are detailed and discussed. Mobile Qualified Server Signature\cite{21} proposes a server side signature creation solution that makes use of a remote HSM to perform signature creation, signing keys are stored outside the HSM which is questionable. Harnessing Electronic Signatures to improve the security of SMS-based services\cite{17} proposes...
a server-side signature creation device that uses some modules offered by the Austrian Government. *Mobile Qualified Electronic Signatures and Certification on demand*[14] is a client side signature creation solution that requires a new approach when issuing Subscriber Identity Module (SIM) cards to users. *A Secure and Flexible Server-Based Mobile eID and e-Signature Solution*[16] proposes a solution that is compatible with different application scenarios, it relays on an HSM to store electronic identification of users.

### 3.3.1 Scheme by Orthacker et al.

Mobile Qualified Sever Signature is a server signature based solution where signature creation is triggered by the user[21]. This solution comprises the knowledge of a Personal Identification Number (PIN) and the possession of a registered Subscriber Identity Module (SIM) card. To interact with the system the user must be able to communicate, with the server-signature web application and with the mobile network carrier (using a distinct channel). Signature creation is left to a server side HSM.

On the server side, the users signing keys are stored outside the HSM due to limited storage issues. However, each signing key is protected by the HSM’s master key, $K_{MK}$. This Master key is stored inside the HSM and since it never leaves its premises, the users signing keys are securely exported. The wrapped signing keys are also encrypted using the users public key, $K_{user}^+$ (see Equation 3.1).

$$key \text{ wrap} = \text{wrap}(K_{signing}, K_{MK})$$

$$\text{encrypted key wrap} = \text{encrypt}(\text{key wrap}, K_{user}^+)(3.1)$$

The private key corresponding to this public key, $K_{user}^-$, is encrypted using the derived user PIN, $K_{pin}$ (see Equation 3.2). Both the encrypted key wrap and encrypted user key are stored in the server’s database along with the users mobile phone number as index.

$$K_{pin} = \text{derive} (PIN)$$

$$\text{encrypted user key} = \text{encrypt}(K_{user}^-, K_{pin})(3.2)$$

When a user requests a signature creation, it is prompted for the PIN through the web interface. When the user inserts the correct PIN, it receives a Short Message Service (SMS)
containing an authorisation code. This code must be inserted in the web interface, only then
does the service create the desired signature. Internally, the service derives the PIN, decrypts
the users private key, and decrypts the wrap with the user’s signing key. The server sends the
wrap to the HSM along with the identification of the user who wishes to create a signature and
the message to sign.

3.3.2 Scheme by Zefferer et al.

This next solution relies heavily on the use of SMS technology to perform transactional
procedures remotely[17]. It includes the signing and delivery of electronic documents.

This solution makes use of some open source modules designed for advanced signature
creation, made available by the Austrian Government, e.g. the Module for Online Applications
- Server Signature (MOA-SS) to integrate signature creation devices for qualified and advanced
signatures.

Figure 3.1: Architecture of the solution proposed by Zefferer et al.[17]

The system, illustrated in Figure 3.1, relies on the use of a central web application to
implement the system’s main functionality. This central element makes use of other components,
such as the SMS gateway to translate SMSs into Extensive Markup Language (XML) web
requests. Qualified signatures are carried out with the help of the Austrian Mobile Phone
Signature and advanced signatures are provided with the aid of MOA-SS, internal to the web
application. Users may also choose to interact with the system through a web browser to keep

\[ a \text{ public key is represented by } K^+ \text{ and a private key is represented by } K^- \]
track of previously signed documents via SMS or in case of administrator users, maintenance tasks.

When a user wants to receive a specific document properly signed, it sends a well-defined SMS. The content of the SMS depends on the procedure being used. A procedure defines the type of document that will be signed and made available to the user. The procedure includes the list of keywords to be used in the SMS request (for signed document creation), an identifier and the intended recipients of the document. This SMS also contains the information whether the created signature will be advanced (only) or qualified, i.e. if it’s going to be signed by the MOA-SS or by the Mobile Phone Signature. New procedures can be defined manually by administrative users, through the Web interface. A user is only a valid user if it is registered in the service. The registration is done via web interface. Once the procedure is completed, the document is sent to the recipient and the user receives a notification by SMS, the file is stored for later inspection.

3.3.3 Scheme by Rossnagel

This mobile device based solution performs qualified signatures on the users mobile by extending the use of the Subscriber Identity Module (SIM) to fulfil the requirements of [2] on Secure Signature Creation Device (SSCD)[14]. To accomplish this, the mobile network carriers must issue SIM cards that can act as a Certification Authority (CA), which according to [14, Rossnagel (et al.)] is not realistic. One way to ease this is by issuing SIM cards with signature capabilities but leave the certification services to some other trusted third-party. For this to happen the user must choose a CA that issues a certificate for the public key stored in the SIM card, so that it can be distributed.

The proposed solution includes a protocol called Certification on Demand. The user receives a SIM card, with a root CA certificate and the ability to generate key pairs, with signing capabilities. The user must ask a CA to issue a certificate on its public key. After the CA issues a certificate, the user can verify the validity of the certificate by checking the signature it contains, using root CA certificate. From that moment on, the user can sign data locally with help of this specific SIM card.
3.3. EXISTING SOLUTIONS

3.3.4 Scheme by Rath et al.

As a counter measure to the problem of signature creation solutions that do not recur to smart cards which are too particular to the environment they were designed for, Rath et al.[16] propose an improved server based solution which is flexible and maintains its compatibility to different application scenarios. It uses an HSM to store all users Electronic Identification (eID) and to create electronic signatures. The requirements for this solution is the flexibility regarding external components (such as which CAs or identity databases that are used), avoid dependency of external entities for authentication (e.g. leaving the user authentication to the mobile network carrier) because it reduces user acceptance, Usability in terms of avoiding the user having to install extra software, and platform independence. The architecture of the solution consists of two parts: the inner part and the outer part, where each part relies on a database. This decoupling allows for a reduction in the impact of data loss in case of an external service gets compromised because the inner part is more limited providing only a few well-defined commands, made available through an encrypted channel.

There are three possible operations: registration, activation and usage. In the registration, the identity of the user is verified. The verification may be by presenting the ID face-to-face to a trusted third party (such as Government entities), by presenting the smart card in the citizen card of the user. Once the registration is complete, users can run the activation process to create their eID, and to associate it to the users’ mobile phone. Activation codes are sent to the mobile phone so that the user can complete the activation process. A signing key pair is then generated inside an HSM. The private signing key is stored outside the HSM, so before being returned by it, the signing key is protected using hybrid encryption (a mix of symmetric and asymmetric encryption). The wrapping of the signing key is based on a signature password which is known by the user.

The certificate containing the public key, the wrapped private key and the eID are stored encrypted in the database. This encryption is based on a password the user chose during the activation process. The usage process is carried out when a user needs to authenticate or to create a signature. To do this, the user must send its mobile number and the password to authorise the operation (this is done by interacting with the user where activation codes send to its mobile phone). After the user authenticates, the encrypted data of the user is decrypted and the (wrapped) private key of the user is sent to the HSM for signature creation.
3.3.5 Comparative Analysis

<table>
<thead>
<tr>
<th>Solution</th>
<th>Signature Creation at Server Side</th>
<th>Signature Creation at Client Side</th>
<th>Smartcard dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheme by Orthacker et al.</td>
<td>×</td>
<td></td>
<td>×</td>
</tr>
<tr>
<td>Scheme by Zefferer et al.</td>
<td>×</td>
<td></td>
<td>×</td>
</tr>
<tr>
<td>Scheme by Rossnagel</td>
<td></td>
<td>x</td>
<td>×</td>
</tr>
<tr>
<td>Scheme by Rath et al.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: Comparative table of Signatures Creation solutions

Table 3.1 summarises the most relevant features of the mentioned solutions. When signature creation takes place at the server side, it is implicit the use of an HSM in all the presented solutions of that type.

The proposed solution by Orthacker et al.[21] does not require the user to have extra hardware to create signatures and since the PIN is not stored in the server (PIN is primarily used for mobile network carrier authentication) and considering that the PIN is essential for signature creation the solution comprises the requirements for qualified signatures. However, since this solution relies on the mobile network carrier for authentication there may be some lock-in in using its services to confirm user’s identity and even the possibility of the mobile network carrier going rogue. This means that authentication of the user might be compromised. Furthermore, regarding the protocol for protecting signing keys, for each user there must be two encrypted objects: one that is the protected signing key and another which is the protected user private key that is used to protect the signing key. Because of this, the storage is not optimised nor performance is ideal because the server has to decrypt private keys before unwrapping the signing key. In terms of the security of this solution, although the HSM wraps the signing key using its master key, it is not stated how the key that encrypts the key wrap is generated. Regardless, the protection of the signing key is not performed entirely by the HSM which does not take advantage of the security potential of this type of devices.

Although [17] presents a smartcard dependent solution, it makes use of a widely used technology, SMS, which allows users to send signed documents using their mobile phone. It does not use the network carrier as a trusted third-party for authentication which, comparing with [21], is an advantage. However, there are some security concerns regarding this solution. For example, when a user wants to obtain a qualified signature, the SMS must contain its password. This password is written in plain text, which is very prone to attacks, specially if past SMSs
containing the password are not deleted. Also, there are also several security issues regarding the Global System for Mobile (GSM) (which is a standard followed by the cellular network regarding the SMS service). One final possibility is the SMS gateway which may be malicious and attacks the system. It is also a tailored solution because it depends on Austrian specific modules and components, which hinders the flexibility of the solution.

The protocol Certification on demand[14] allows scalable signing functionality because of the widespread usage of SIM cards on mobile phones. Besides that, the user does not need to carry a mobile phone equipped with a Secure Signature Creation Device (SSCD) on an extra slot, rather the user can use its SIM card and even decide not to use the signing capabilities of the cards if it does not need them[14]. For the signature creation take place at the client side there is also no message overhead to create signatures. However, this solution requires a great deal of changes in the mobile carrier’s infrastructure which implies a business dependence in the solution. There is an bigger infrastructure lock-in in this solution when compared with [21] and [17]. That is the cost of having client-side signatures. There is also the issue of battery usage in signature creation taking place at the client side which means that this solution is more demanding on the computational power of the device.

Rath et al.[16] present a solution which is flexible to different components that support such a system. It allows several options of registration and usage. Signing keys are generated in the HSM and stored in a database. What this solution lacks is the conditions in which data is exchanged with external entities. This is specially important, because this solution trusts these entities for user identification.

What can be concluded of this analysis is that, even though platform independence and convenience is possible due to recent Regulation eIDAS[2], existing server-side solutions rely on extra hardware at the client-side in order to accomplish remote qualified digital signatures. This is because there needs to be a secure link between the user and the components that provide qualified signatures which is not the same as being connected to a service that provides qualified signatures. In order to provide a distributed qualified signature creation services, components such as web servers in public networks are made available to client applications to serve as intermediate to inner components. These components cannot be fully trusted by users and the components that directly create the qualified signatures (such as HSMs).

Many of the remote signature solutions require that the user possesses a SIM card to com-
municate with it, meaning that the user must be bound to a (usually country specific) mobile network. This does not help to accomplish the eIDAS goal of having service homogeneity in EU countries services. There is a gap in the market for solutions which are not dependent of specific third-party components and that provide a convenient way for users to obtain qualified signatures without having to carry extra hardware.

3.4 Summary

There are many solutions proposed as a consequence of the EU Regulation 1999/93/EC[1]. These solutions rely heavily on the use of SIM cards. That is because of the restrictions of the signature creation taking place at the server side which requires the user to be authenticated. Since EU Regulation 910/2014[2] these restrictions are more relaxed.

Solutions can be divided in server and client-side. Client-side solutions are usually hardware dependent whereas, in theory, server-side solutions allow users to digitally sign data without needing extra hardware. However, the latter is not true for the existing solutions, they usually rely on the use of SIM cards for user authentication which, besides rendering them platform dependent, requires the third-party mobile network to be trusted. Furthermore, some of these solutions are supported by external modules or components for remote/server signature creation which are limited to different countries.

There are not many that provide hardware independent, secure and flexible ways of creating qualified signatures, giving the opportunity to innovative solutions to be proposed.
The ability to perform operations using a device which is connected anywhere presents convenient possibilities for users and their daily online activities. Qualified digital signatures provide data authenticity which allow for users to carry out these activities in a more secure manner. On one hand, this benefits the user because he can securely interact with the service layer; on the other hand it benefits the service layer since it can trust that the users are authentic. Qualified signatures can only be created with the users’ permission and exclusively for the data they authorise. The requirements for qualified signature creation are defined in eIDAS[2] and the goal is to make them verifiable by different EU services.

Towards the creation of qualified signatures, several solutions[21][17][14][16] have already been proposed to solve this problem. In Chapter 3, some issues were identified regarding these solutions. Many of them rely heavily on SIM cards for user authentication which implies a hardware and third-party dependency in order to accomplish remote signing functionality. This presents interesting opportunities for new solutions that overcome these issues to be proposed, such as the one in this document: RSIGN.

The proposed Remote SIGNature (RSIGN) solution consists of a remote HSM-based system (Requirement 4) that allows users to securely obtain remote qualified signatures without the need of carrying any other hardware, besides their mobile device (Requirement 1 and Requirement 2). In order to authorise signature creation and being compliant with eIDAS[2] (Requirement 5), a given user must correctly input his/her PIN to explicitly authorise the operation. The strength of this solution is in 1) the secure and scalable way the users’ signing data is managed, and 2) the use of a secure channel to protect exchanged the sensitive data between the user endpoint and the service, against a great deal of attacks to the Authenticity, Confidentiality and Freshness of the data (Requirement 3). This solution protects the sensitive data even if the attack is performed from the internal infrastructure because only a few components have access to the data in plain text.
RSIGN is part of a broader system, in which users can use a myriad of services. Particularly, users are able to choose alternative ways to have a certificate which imply the generation of a private key that can be used to create digital signatures. For example, the user may choose to have token-based solution where signing happens at the client-side.

RSIGN was created in collaboration with Multicert[22], a company that provides digital certification and security solutions. This is relevant because, besides Security, issues such as flexibility and scalability were also taken into account throughout the design phase.

Once again, the goal of signatures is to authenticate data. The way to verify the authenticity of the data is trough the signature verification process. RSIGN is not concerned with this verification. This is because the goal of qualified signatures is to have other services, even if one of Multicert’s, responsible for verifying its security properties (so that they can carry out other operations). However, signing key pairs are generated in RSIGN’s HSM and for that reason, public keys, that can be used to verify the signatures created by RSIGN, must be exported after they are created so that they can be externally certified.

The platform the user is using to interact with RSIGN (e.g. a smart phone) is different from the human user itself. Accordingly and throughout the remaining of this thesis, the platform will be mentioned as the user endpoint or client application. This means, for instance, that is not actually the user that is establishing a secure channel with the service, but rather that the client application is doing so; it is also not the user endpoint that inputs the PIN to authorise signature creation, instead it is the user himself/herself.

This chapter starts off by presenting the Trust model (Section 4.1) stating some premisses considered during the design of the solution. Then an overview of the RSIGN’s architecture (Section 4.2) is explained, including its interaction with other Multicert components, and their importance for RSIGN. After that, key aspects of the solution are further detailed, as they are the core of the solution (Section 4.3, Section 4.4 and Section 4.5). Then RSIGN is detailed in terms of functionality (Section 4.6) for the operations available for client applications to use.

### 4.1 Trust Model

Since the solution herein presented does not prevent against the infinite type of attacks there is, but they are not to be ignored, this section presents some premisses made before and
during the design of the RSIGN. They will be relevant when discussing the security features of the solution.

- **User data privacy:** credentials and PINs are kept secret by the user owning them. The user authentication and authorisation rely on it;

- **Security of the user endpoint:** It is assumed that the client application is running in a non-malicious platform and the certificates’ validity is verified regularly;

- **Client-side Hashing:** The platform implementing the User Interface (UI) must be capable of securely generating hash values for the data to be signed;

- **Secure Qualified area:** The components in the Qualified area (RSIGN Backend and HSM) are robust and supported by certified hardware, and accessed in a way that they cannot be compromised into performing malicious actions themselves;

- **Users’ authentication:** The user authentication, i.e. login operation, is assumed to be securely performed by the Client Portal;

- **Signing keys storage:** The Multicert (MTC) Backend is honest but can be curious. It retrieves the protected signing when needed for the corresponding user, Data Authenticity is assured;

- **TLS data:** The certificates and key pairs used to authenticate entity communication through TLS are kept safe and their validity is verified regularly;

- **The system is available:** Mechanisms to prevent attacks to the system’s availability, e.g. flooding of the Client Portal, are assumed to be in place;

- **Cryptographic algorithms’ soundness:** The cryptographic algorithms used by the HSM, the RSIGN Backend and the client application (in establishment and usage of the secure channel) are sound, i.e. not breakable by cryptanalysis.

With these assumptions, the emphasis on the core aspects of the solution prioritization relaxes some other. The security of the endpoints (client application and Qualified area components), in which the secure channel and the protection of signing keys rely on, being the strongest assumption.
4.2 Architecture Overview

The architecture, depicted in Figure 4.1, presents the overall components of RSIGN. On the client side, the user accesses the service through its mobile device, such as a smartphone or a laptop, connected to the internet. Operations triggered by users are redirected to the Client Portal, which includes the frontend of the system. This component is responsible for authenticating the user in the system, through its email and password. If successful, it redirects requests and responses to/from the MTC Backend.

The MTC Backend is responsible for storing signing keys, generate the associated certificates, and making requests to RSIGN when necessary. The RSIGN Backend receives requests and serves as the application logic supporting HSM-related operations. As mentioned before, the HSM performs the needed cryptographic operations and stores sensitive data (Requirement 4).

For simplicity’s sake the Client Portal and the MTC Backend were not fully displayed in this initial figure. Both of these components are, in fact, composed of several components that, together, allow the desired functionality. The actual components, illustrated in Figure 4.2, are: the Client Portal Frontend, which makes the service available to client applications, the Client Portal Backend which deals users’ authentication by verifying their credentials against what is stored in the Authentication Database. The MTC Backend is composed by the RADIX, a component that manages the users’ certificates and is responsible for redirecting requests/responses to/from RSIGN. It relies on a database used to store users’ certificates (Certificate Database) and a database to store signing keys (Key Database), once they are registered in RSIGN.
4.2. ARCHITECTURE OVERVIEW

Even though these components are part of the solution, they are services already deployed in the Multicert’s infrastructure that support RSIGN, which means that they are not the core of the solution. Throughout this thesis, the components may be referred to as their abstraction (Client Portal or MTC Backend). For instance, instead of stating that the RADIX requests the Key database for a signing key, it is just mentioned that the MTC Backend retrieves the given signing key.

In order to have a flexible system to be integrated in the existing Multicert’s infrastructure, different components are used to provide remote qualified signature creation. These components are separated in areas based on function and security requirements. For instance, the HSM is placed in the most secure and internal area, the Qualified area, whereas the Client Portal’s frontend is placed in a less secure area. Given this separation, not all of these components are trusted equally. Because of that, Security measures have to be taken accordingly, e.g. the Client Portal and the MTC Backend cannot get access to users’ PINs because they are used by other services, hence they are not in the Qualified area which is limited to qualified signature creation.

In RSIGN, the communication between the components that are web applications (Client Portal Frontend, Client Portal Backend, MTC Backend and RSIGN Backend) is made through Representational State Transfer (REST). REST is an architectural style based on lightweight communication (HTTP). It allows for components in RSIGN to communicate in a simple way through well-defined Universal Resource Locators (URLs). REST also allows for the solution to be implemented using different platforms as the client application, due to its flexibility and widespread usage.
4.3 Export Signing keys

For this solution, a HSM was used to remotely create qualified signatures (Requirement 4). Usually, HSMs have limited storage capabilities. This means that, in order to have a solution that is not limited to particular HSMs and is scalable in terms of the number of users it supports, signing keys are exported and stored outside the HSM. Specifically, they are stored in the Key Database. These signing keys are identified in the database by the certificate issued under the corresponding public key. This way, the system’s scalability is not limited to the number of signing keys the HSM can store.

Since the signing keys, which are ultimately used by the HSM to create qualified signatures on behalf of the users, are stored outside its secure environment (in a database outside the Qualified area), they must be protected in such a way that only the HSM can use them, otherwise it cannot be stated that the solution provides digital signatures with the qualified status. Furthermore, in order to fulfil Requirement 5 on compliance with eIDAS[2], RSIGN must prevent that none other than the user, owner of the signing key, can authorise signature creation. In RSIGN, this is accomplished by protecting a user’s signing keys with the HSM’s master key (only known by it) and the user PIN (only known by he/she). Moreover, this PIN is not persistently stored in the service which means that the user authorisation is required in order for RSIGN to obtain the signing key and create qualified signatures. This aims to attend the requirement for qualified signatures "it is created using electronic signature creation data that the signatory can, with a high level of confidence, use under his sole control;"[2].

When a user gets registered in RSIGN, the HSM generates a key pair that can later be used to sign and verify data’s authenticity. It also generates a PIN, preferably at least four digits long, that can later be used to authorise signature creation.

Before returning the public key to be certified (out of RSIGN’s environment), the RSIGN Backend requests the HSM to protect the private signing key. Although the HSM’s master key is used in this step, it is not used directly to perform a key wrapping operation on the key (described in Equation 4.1).

\[
K_{\text{wrapping}} = \text{derive}(K_{MK}, \text{hash}(PIN))
\]

\[
key\ wrap = \text{wrap}(K_{\text{signing}}, K_{\text{wrapping}})
\]

(4.1)
The master key, $K_{MK}$, is used to derive a wrapping key, $K_{wrapping}$. As a parameter to the key derivation operation, the hash of the newly generated PIN, $\text{hash}(PIN)$, is used. The hash of the PIN is used instead of just the PIN to obtain a stronger key.

$K_{wrapping}$ is then used to wrap the recently generated signing private key, $K_{signing}$. The resulting key wrap is returned to the MTC Backend and stored in the Key Database. These operations that occur inside the HSM are displayed in Equation 4.1.

The RSIGN Backend interacts with the HSM through a PKCS#11 interface. This way the solution does not depend on a proprietary or specific HSM.

The users’ key pair, corresponding wrapping keys and the PINs exist in the HSM only while a PKCS#11 session is open, i.e. they are session objects. This means that only the HSM’s master key is persistently stored and that, apart from when it is being used, the users’ private signing key only exists in a secure wrap. It can only be unwrapped with both the user’s PIN and the specific HSM. Furthermore, the master key is not directly used to wrap signing keys which means that there is not enough information for the HSM to obtain the signing key and create qualified signatures, not without the user’s PIN.

To fulfil Requirement 5, ideally not even the RSIGN Backend would have access to the generated random PINs in plain text, i.e. the trusted endpoints should be the HSM and the user endpoint. However, PKCS#11 does not allow the generation of random values and the internal protection of those values before returning them. For this reason, and because the RSIGN Backend is a trusted component in the Qualified area, PINs are returned to the Backend in plain text after their generation. Before including the PIN in the registration response, though, they are protected and are not accessible (in plain text) by components in between the Qualified area and the user endpoint.

There is a case where the protection of signing keys does not favour the internal usability of the system. Usually public keys corresponding to private signing keys are used to issue a digital certificate, as mentioned in Section 2.1. This certificate can later be used to verify signatures created by RSIGN, without it the signature has no great value. Certificates are only valid for a period of time for reasons such as to enforce keys to be certified by up-to-date algorithms. Master keys should also not be valid forever, because if they are compromised, all signing keys that were protected by it in the past are also compromised. For this reason the same good
practice of changing Master keys regularly should be applied.

In RSIGN, if a master key needs to be changed for some reason, all the signing keys that were protected by it, need to be wrapped again using a wrapping key derived from a new master key. To unwrap all the signing keys that were protected using the old master key, the user’s PIN (only known by himself/herself) is needed to derive the unwrapping key based on the new master key and obtain the signing key. Since the changing of the master key is a back-office task and should occur without the knowledge of the users, this presents an usability issue.

An alternative to overcome this problem is to protect a given private signing key with two protection layers (see Equation 4.2): first the PIN is used to derive a key, $K_{wrapping}$, that is then used to wrap the signing key. After that, the HSM’s master key is used to encrypt the previous result. This means that if the master key needs to be changed, only this upper protection layer is undone to encrypt the result again with a new master key. The signing key remains protected and is only obtained for the signing operation.

$$K_{wrapping} = \text{derive}(\text{hash}(PIN))$$

$$\text{key wrap} = \text{wrap}(K_{signing}, K_{wrapping})$$

$$\text{encrypted wrap} = \text{encrypt} (\text{key wrap}, K_{MK})$$

(4.2)

The two-layered result could then be exported out of RSIGN in the same way.

Although this solution offers a more convenient way to protect private signing keys in terms of usability, there are two reasons that motivate the use of the initial approach. The first is that the wrapping key is derived from a weaker input, the hash of the PIN, rather than the master key (recall Equation 4.1). The second is a constraint in the Multicert’s policy for objects protected with invalid master keys. It states that these signing keys should no longer be valid either. The solution favours the integrability of the Multicert’s existing policies in turn of usability.

4.4 Using signing keys

The ultimate goal of this solution is to allow remote qualified signature creation. Internally, this means retrieving the signing key which the user owns, using the certificate associated with
the signing key, from the MTC Internal Key Database. Along with the data to be signed, the
PIN necessary to authorise signature creation is sent, through the secure channel, by the user
endpoint. On receiving the data to be signed, the PIN and the (wrapped) signing key, the
RSIGN Backend requests the HSM to unwrap the signing key, $K_{\text{signing}}$, and use it to sign data,
$data$, reverting what was done in Export Signing keys and then obtain the $\text{signed data}$, as shown
in Equation 4.3.

\[
\begin{align*}
K_{\text{unwrapping}} &= \text{derive}(PIN) \\
K_{\text{signing}} &= \text{wrap}(\text{key wrap}, K_{\text{unwrapping}}) \\
\text{signed data} &= \text{sign}(data, K_{\text{signing}})
\end{align*}
\] (4.3)

4.5 Secure Channels

In order to achieve a secure solution, the communication going inwards and outwards from
each component must be protected in some way. The Transport Layer Security (TLS) is a widely
used standard which provides security properties to the messages exchanged between components
that are directly connected. In RSIGN, servers communicate through these channels to carry
out the expected operations in a secure way. These components are mutually authenticated and
client applications interact with an authenticated server (one-way authentication). Even though
TLS provides Confidentiality, Authenticity and Freshness to the exchanged messages, it must be
assured that users’ sensitive data (in this case PINs, data to be signed, and signatures) exchanged
between the user and the components in the Qualified area is protected against attacker an non-
trusted components, in order satisfy the requirements defined in eIDAS[2] (Requirement 5).

Figure 4.3 illustrates the secure communication channels between the several system com-
ponents, i.e. a TLS connection among communicating components and a secure channel on top
of the TLS channels that is established before any sensitive data is exchanged.

The establishment of the secure channel, results in a link between the user endpoint and
the RSIGN Backend. It is initiated by the Diffie-Hellman Key Exchange (DHKE) protocol and
reinforced by other mechanisms. After the user’s credentials are verified by the Client Portal at
login, the user’s endpoint and the RSIGN Backend agree on a shared secret key. Then, the user
and the RSIGN exchange random data to confirm the use of the agreed shared secret. From
that moment on, the channel is established and both the user and the RSIGN Backend (which by extent is the Qualified area), share a secret key that is used to protect data that not even intermediary components, i.e. the Client Portal and the MTC Backend, can have access to in plain text.

The RSIGN Backend keeps track of the keys it shares with client applications, i.e. it maintains an open channel with them. The key is valid during the user operations in a given session or when a time out is reached, meaning that a new login requires a new shared secret to be generated.

Note that the user endpoint and the RSIGN Backend are not directly linked, the user continues to only interact with the Client Portal and the RSIGN Backend continues to only interact with the MTC Backend and the HSM. This means that the protocol does not provide mechanisms against attacks to Availability. An internal malicious component, e.g. Client Portal, could simply decide not to forward messages that are intended to RSIGN Backend, sent by the user’s endpoint. What this channel aims at is protection against attacks to the Authenticity, the Confidentiality and the Freshness of the data (Requirement 3).

The Diffie-Hellman Key Exchange (DHKE) protocol[23] outputs a shared secret key that provides Confidentiality to the data being exchanged. However, this protocol lacks Data Authenticity and Freshness mechanisms which makes it vulnerable to the man-in-the-middle attack. Therefore, the Hashed Message Authentication Code (HMAC)[24] mechanism, digital signatures, and the use of random nonces, are added to assure the aforementioned security properties.

Even though the shared secret is used to protect messages being exchanged, it is not used directly for either encryption and for the data authenticity mechanism HMAC. This is due to related key attacks when using the same key for both purposes. Instead, for messages going
through the intended secure channel, the shared secret key, $K_{\text{user-rsign}}$, is derived into two symmetric keys, $K_c$ and $K_a$, the confidentiality and data authenticity key, respectively.

This key derivation is done by computing the SHA-256 hash value of the shared secret, $K_{\text{user-rsign}}$, and splitting the result into two halves: the first is used to build the confidentiality key, $K_c$, and the second half is used to build the data authenticity key, $K_a$. Both these keys are not persistently stored, neither at the RSIGN Backend nor at the client application which, in the security point of view, is an advantage. An attacker who got access to a protected message and the shared secret would still need to know how to obtain both $K_c$ and $K_a$ to successfully obtain the message content in plain text or to tamper with the message.

### 4.5.1 Secure channel establishment

Figure 4.4 illustrates the protocol for the establishment of the secure channel on top of the existing TLS channels. Before the protocol is initiated by the client application, it must have access to the RSIGN Backend certificate. The client’s application already contains the certificate used to establish the TLS channel with the Client Portal Frontend (corresponding key pair of type RSA, 2048 bits long). For the establishment of the secure channel on top of the existing, a different certificate is used. This certificate corresponds to a key pair of type RSA, 4096 bit long. The use of keys with this length decreases performance but since this key pair is only used in the establishment of the secure channel, as detailed below, it was given priority to higher security.

**Protocol step 1**  
Previously to sending this message, the user’s endpoint generated the Diffie-Hellman (DH) key pair. The message is sent by the user endpoint to RSIGN Backend. It consists of the user’s DH public value, $Y_{\text{user}}$, and a nonce that is encrypted with the RSIGN Backend’s public key, $[N_1]^K_{r\text{sign}}$, included in the RSIGN Backend’s certificate.

Upon receiving the message, the RSIGN Backend generates its DH key pair and using the newly generated private value and the user’s public value, $Y_{\text{user}}$, computes the shared secret key, $K_{\text{user-rsign}}$. Afterwards, it decrypts the nonce in the request, $[N_1]^K_{r\text{sign}}$, using $K_{r\text{sign}}$.

---

1 A given message $M$ encrypted using a symmetric key, $K$, is represented by $\{M\}_K$ (braces). If the key used to encrypt were a public key, then it would be represented by $[M]_K$ (brackets).

2 A public key is represented by $K^+$ and a private key is represented by $K^−$. 
Protocol step 2  The second message is sent by the RSIGN Backend’s to the user’s endpoint. It contains the RSIGN Backend public value, $Y_{rsign}$, that is going to be used by the client application to generate the shared secret key, as defined by the DHKE[23]. It can be sent in plain text because an attacker would also need the client’s private value to calculate the shared secret, $K_{user-rsign}$.

Along with the RSIGN Backend’s public value, $Y_{rsign}$, this response message contains the response nonce, $N_1 + 1$, and the signature of the response nonce, $\text{signature}(N_1 + 1, K_{rsign}^-)$ both encrypted using $K_c$, the Initialization Vector (IV) used for encryption and the hash value of both the encrypted data plus the IV, $IV||\{\text{signature}(N_1 + 1, K_{rsign}^-)||N_1 + 1\}K_e$, calculated using the data authenticity key $K_a$.

On receiving the message, the client application computes the shared secret key, $K_{user-rsign}$ using $Y_{rsign}$ and calculates both confidentiality and data authenticity keys, $K_c$ and $K_a$, respectively. Then it uses $K_c$ to decrypt $\{\text{signature}(N_1 + 1, K_{rsign}^-)||N_1 + 1\}K_e$, $K_{rsign}^+$ to verify the signature, $K_a$ to verify the hash value and the nonce in the request to verify its increment in the response.

Since only the RSIGN Backend knows how to decrypt $N_1$ sent in message 1 (using private key $K_{rsign}^-$), only it could respond with the response nonce and it’s signature value. Plus authenticity of the response is guaranteed by the HMAC value sent along the message. This also includes
4.5. SECURE CHANNELS

the IV authenticity (confidentiality of the IV is not required).

**Protocol step 3 and 4** Now that both the user and the RSIGN share a secret key, $K_{user-rsign}$, a confirmation step follows. It consists on the user sending a protected nonce and the RSIGN backend responding with the incremented nonce. On success, both the user and the RSIGN Backend know that the other entity can successfully receive and respond to messages through the newly created channel.

### 4.5.2 Secure channel usage

After the secure channel is established, messages can be exchanged in the following way: the sender that wishes to send a message, $M$, derives an encryption and integrity keys, $K_e$ and $K_a$, from the agreed shared secret key, $K_{user-rsign}$, and computes the HMAC of the encrypted data plus nonce, $N_1$, to be sent. It then sends the encrypted message and the HMAC value alongside it (see (4.4)).

$$K_e, K_a = \text{derive}(K_s)$$

$$A = \{\text{data}, N_1\}_{K_e}$$

$$M = A, \text{HMAC}(A, K_a) \quad (4.4)$$

The receiver then derives the encryption and integrity key (the same way the sender did) and computes the HMAC value of the encrypted message to verify the authenticity of the data. After decrypting and processing the data, it responds to the sender with a new message (depending on the operation), $M'$, containing response data, *response data*, and the incremented *nonce*, $N_1 + 1$ (see (4.5)). If the original sender verifies that the message is authentic and the nonce is as expected, it accepts the response and processes it.

$$B = \{\text{response data}, N_1 + 1\}_{K_e}$$

$$M' = B, \text{HMAC}(B, K_a) \quad (4.5)$$
4.5.3 Alternative approach

Alternatively, a different protocol could have been used to provide the same security properties: Confidentiality, Data Authenticity and Freshness. Instead of the user and the RSIGN Backend agreeing on a shared secret key through the DHKE protocol, the user endpoint would generate a key pair locally, $K_{user}^+$ and $K_{user}^-$ the public and private key respectively. Then, using the public key ($K_{rsign}^+$) in the certificate it has access to, the client application would wrap the generated public key and send the result to the RSIGN Backend along with a nonce encrypted using also the RSIGN Backend public key included in the RSIGN Backend’s certificate, $K_{rsign}^+$. When the Backend receives this message, it is able to decrypt both the key, $K_{user}^-$ and the encrypted nonce, using its private key, $K_{rsign}^-$. To confirm towards the user endpoint that the RSIGN Backend did in fact receive the message, the user endpoint expects that the response message contains the incremented nonce, $N_1 + 1$, encrypted with $K_{user}^+$. Once the channel is established, authenticity of the data would be assured to following message using digital signatures.

The amount of computation using this alternative would be reduced, because the DHKE protocol would require generating key pairs in both sides before calculating the shared secret, instead of just generating a key on the client-side. However, with DHKE the key is never sent through the network, making it more secure. Since this RSIGN aims at protecting data through this secure channel, it was given priority to security over performance.

4.6 RSIGN’s functionality

After a description of the system’s architecture and some key aspects of the solution, the following section will detail the operations available to client applications.

Figure 4.5 displays a web page that already exists for qualified certificates purchase\[25\] extended to include the possibility to obtain qualified signatures remotely.

The main goal of the solution is to create qualified signatures (sign operation) in the name of a given user. Before that, the user must be registered in RSIGN. During registration, a signing key pair is generated inside the HSM, then wrapped and then exported. Also generated by the HSM, is a PIN that is delivered to the user, through the secure channel.
4.6. RSIGN’S FUNCTIONALITY

Figure 4.5: Part of Multicert’s web page with remote qualified certificate available.

Later on, the user may also wish to change his/her PIN. This is important because the user may want to authorise signature creation using a PIN that is easier for him/her to remember. Regardless, the PIN is only known by the service during the operation and is only known in plain text by the components in the Qualified area, since it is sent through the secure channel.

These features were made available to the client via a REST API. The client application connects to the Client Portal to use the service. It authenticates the Client Portal through TLS. Given the internal functionality that is happening, each command sent by the client application may translate in commands made to RSIGN, which also interfaces with MTC internal through a REST interface. For this, the MTC Backend and the RSIGN Backend must be mutual authenticated through TLS (recall Figure 4.3).

4.6.1 Signup

This operation allows the user to become a Multicert user. Through this operation, the user can use several Multicert services, including RSIGN. This signup is not the core of the solution this document presents, but it is necessary to use services in the existing Multicert infrastructure. Nonetheless, in terms of security, it provides a user authentication that prevents unauthorised users from making requests to more internal areas.

To signup, the user must provide his/hers email and password. The email and a hash of the password are stored in the Client Portal’s database. The Client Portal verifies if there are already a user signed up under the same email, to avoid ambiguity.
4.6.2 Login

After the signup, this login operation could be as easy as having the Client Portal verify the credentials provided (email and password) against the credentials it has stored, without any involvement of inner areas’ components. However, in the case the user chose to use RSIGN and as stated before, it is at login that the secure channel, that allows the user’s endpoint and the RSIGN Backend to interact in a secure manner, is established. This means that after verifying the credentials, the protocol described in Secure Channels takes place. On success, the user is authenticated, and he/she can use RSIGN.

4.6.3 Registration

The following operation, depicted in Figure 4.6, represents the registration in RSIGN, i.e. signing key pair and PIN generation inside the HSM, necessary for the signing operation. It also includes what happens at the MTC Backend in order to later retrieve the exported wrap containing the signing key within (Signing and Change PIN operations). This operation depends on the Login operation because there is sensitive data that is going to be exchanged through the secure channel between the Qualified area and the user endpoint.

Figure 4.6: Sequence diagram presenting user registration in RSIGN
In this diagram, the user requests its intention to be able to sign data in the future. For that, a key pair and a PIN are generated by the HSM. Then, the private signing key is wrapped, just as described in Exporting Signing Keys. The corresponding public key, the PIN and the wrap are returned to the MTC Backend.

On receiving the message, the MTC Backend issues and stores a certificate under the mentioned public key. It then returns the message containing the PIN to the Client Portal. The Client Portal delivers the protected PIN to the user only if all the previous steps are successful. Registration in RSIGN is complete.

Note that the PIN is not sent in plain text, instead it is sent through the secure channel that was previously established between the RSIGN Backend and the user endpoint. Outside the Qualified area, only the user’s endpoint can have access to the PIN, in plain text. The channel guarantees the Confidentiality, Authenticity and Freshness of the PIN.

4.6.4 Signing

This operation, illustrated by the diagram Figure 4.7, requires that the user is already registered in RSIGN. It also requires that the user is authenticated in the Multicert service through login, to ensure that a secure channel between the Qualified area and the user’s endpoint exists.
In the signing request, the client application can send more than one data object to be signed. This is a feature motivated by a usability requirement. Instead of the user having to insert the PIN for every data object it wishes to obtain the qualified signature of, it can batch several data objects in the request. However, the client application does not send the data objects in plain text. When building the request, the client application calculates the hash value of each data object which is actually the only thing the HSM needs to encrypt with the user’s signing key. This avoids remote entities, authorised or not, from getting access to the contents of the data objects.

When the MTC Backend receives a signing request, it retrieves the wrap containing the user’s signing key, using the certificate information. Then it forwards the protected PIN and data to be signed to the RSIGN Backend. The interaction described in Using Signing Keys takes place inside the Qualified area. If successful, the result is sent in the response message to MTC Backend. Note that this data is sent through the secure channel, meaning that intermediate components will not know of the signature value of the data that was initially requested to be signed.

### 4.6.5 Change PIN

Users may request the service to change their PIN. This requires that, assuming it is registered in RSIGN and authenticated through log in the service, the (wrapped) signing key is retrieved from the MTC internal database, key wrap, and together with the old and new PIN, respectively old PIN and new PIN, sent to the RSIGN Backend. Internally, the interaction described in Section 4.3 is undone inside the HSM and, if successful, repeated, only this time with the new PIN. This operation is displayed in Equation 4.6.

\[
\begin{align*}
K_{unwrapping} & = \text{derive}(K_{MK}, \text{hash(} \text{old PIN} \text{)}) \\
K_{signing} & = \text{unwrap(key wrap, } K_{unwrapping}) \\
K_{wrapping} & = \text{derive}(K_{MK}, \text{hash(} \text{new PIN} \text{)}) \\
\text{new key wrap} & = \text{wrap}(K_{wrapping}, K_{signing})
\end{align*}
\] (4.6)
4.6.6 Log off

The log off operation prevents the user’s endpoint from abruptly leaving the service. This means that on log off, the RSIGN Backend, which has no way of knowing when users have left the service, gets notified that a given user is logging off. The secret key that the RSIGN Backend shares with the user is discarded which means that there is not secure channel between the RSIGN Backend and the user that wishes to leave.

4.7 Summary

In order to provide a solution that accomplishes qualified signatures, i.e. compliant with eIDAS\[2\], and to overcome the disadvantages identified in existing solutions, this chapter presents the proposed Remote SIGNature (RSIGN) architecture.

RSIGN is part of a broader service that Multicert\[22\], a company providing digital certification and security solution, provides. It allows for users, using their mobile device, to sign data, in a qualified manner, without needing extra hardware. It contains a Hardware Security Module (HSM) as the cryptographic device that creates the signatures in the name of the user. Because HSMs usually have limited storage capacity and to allow the solution to be implemented using different HSMs, users’ signing keys are stored externally.

eIDAS states that only the HSM can use the users’ signing keys, for that reason, these keys are protected using a master key that is only known by the HSM. eIDAS also states that only when the users’ want to, can qualified signatures be created in their name. To satisfy this requirement, the users’ PIN (also generated by the HSM) is used to, together with the HSM master key, protect the signing key. This way, only with both user and HSM involvement can signatures be created.

The way data is exchanged between the user and the component supporting the HSM, RSIGN Backend, needs to be protected against intermediate components. For that, a secure channel is established. This channel provides Confidentiality, Data Authenticity and Freshness to sensitive data, such as the PIN the user has to send in the signing request.

Both the RSIGN Backend and HSM are stationed in a Qualified area, with restricted access from outside components, because they have access to sensitive user information.
To be able to sign, the user must first get registered in RSIGN. The (wrapped) private signing key, the public key matching it, and the (protected) PIN are returned by the Qualified area. Internally, Multicert’s components generate a certificate for later signature verification, store the wrapped signing key and deliver the protected PIN to the user. The user may also change its PIN, for that the wrap protecting the signing key of that user must be changed. It requires the HSM to unwrap the signing key and wrap it with the input of the new PIN.

RSIGN provides a convenient and secure way to obtain qualified digital signatures to data that users, remotely connected through their mobile device, intend to sign. Its main advantages is the way users’ keys are managed as well as the protection of the transmission of sensitive data between the user and the service, which makes it compliant with eIDAS[2] requirements.
5

RSIGN’s Implementation

There are several components in RSIGN’s architecture. Together they interact to provide the desired functionality. To accomplish this, several technologies were used. This chapter will state implementation details of the solution.

RSIGN is supported by components that are deployed in the existing Multicert infrastructure. Since they are not the core of the solution, they were not actually used. Instead, and to present a complete solution, their functionality was simplified (Section 5.1).

In terms of the implementation, main programming language, RSIGN was also implemented in Java, to ease the integration of modules in the existing Multicert’s infrastructure. Java is also known by the frameworks it has available. From building tools to database communication, details of their integration in RSIGN will herein be stated (Section 5.2).

PKCS#11 was chosen for the interaction standard with cryptographic devices, such as HSMs. The development of RSIGN started with testing the feasibility of the core HSM tasks to protect users’ signing keys (explained in Section 4.3) using this interface. Because HSM PKCS#11 implementations are in the C language, PKCS#11 wrappers are used to translate other language commands in PKCS#11’s. In RSIGN, the IAIK PKCS#11 wrapper[26] is used to translate Java commands intended to communicate with the system’s HSM.

Key wrapping is a common command while using HSMs. However, this solution requires that the PIN or rather the hash of the PIN is used as a parameter to a PKCS#11 derivation mechanism. Key attributes are defined in the key templates that are used as parameters to their generation, particularly the unwrapping and the key derivation. The specific PKCS#11 attributes for each type of key in RSIGN and the algorithms used are further detailed in Section 5.5.

Mobile applications are in our everyday life, technology wise. For that reason, and regarding the client side, this UI presents a commercially and attractive way to show how RSIGN can be
used to obtain remote qualified signatures (Section 5.6) while users are interacting with their Android mobile device. The interface is of simple design and, in few steps, a regular user can obtain qualified signatures for documents they have stored in the device.

5.1 Prototype Implementation

Chapter 4 presents a solution with emphasis on the Qualified area and the secure channel between the user endpoint and the RSIGN Backend. To have a product that considers the whole use case, from the user to the HSM, some components were simplified to accomplish the desired functionality, i.e. the Client Portal and the MTC Backend. This was necessary because the deployment of RSIGN was not possible in the given time period of the development of the solution. The functionality that was simplified, although still implemented, was the user authentication in the Multicert system and the certificate generation for the users’ signing key pair. Figure 5.1 represents the architecture of the implemented prototype of Multicert infrastructure with RSIGN within.

<table>
<thead>
<tr>
<th>MTC Internal area</th>
<th>Qualified area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate Database</td>
<td>Keys Database</td>
</tr>
<tr>
<td>Authentication Database</td>
<td>Client’s Portal</td>
</tr>
<tr>
<td>HSM</td>
<td>RSIGN Backend</td>
</tr>
</tbody>
</table>

Figure 5.1: Implementation architecture

The Client Portal acts as both the Client Portal Frontend and Backend. It is still this component that interacts with the Authentication database (which stores the users’ credentials) but it also generates certificates for the signing key pairs. The RSIGN Backend still communicates with the HSM but it also receives requests from the Client Portal and stores the signing keys in a Key Database. The idea is that the RSIGN manages the signing keys during their whole lifetime, without the Client Portal having knowledge of them in any way. This makes RSIGN deployable as a more complete service meaning that the components using RSIGN need to do...
less, since it provides signing key management.

Since it is the RSIGN Backend that manages the signing keys, it does not know if, in the user registration after it generates the signing key pair, the certificate was successfully generated. If something goes wrong, signing keys should not be available for signing because there is no certificate to verify them. For that reason, after the RSIGN Backend returns the PIN and the signing key pair, it waits for a *activation notice* stating that the certificate was generated. The sequence of steps in RSIGN registration are as follows:

1. The Client Portal requests RSIGN a new user registration;
2. The key pair and PIN for the user is generated and the key pair is properly stored;
3. The public key and the PIN are returned to the Client Portal;
4. The Client Portal generates a certificate for the given public key;
5. The Client Portal sends an activation notice to RSIGN;

This way, RSIGN knows that signing keys are ready to be used in later signing requests. The signing key status on the Key Database is updated after the activate notice is received. Only then the user registration in RSIGN is complete.

## 5.2 RSIGN’s Technology

RSIGN consists of a web service application offering a REST API for client applications that intend to allow users to use the remote service.

RSIGN’s logic was implemented in Java[27]. This programming language is widely used in many applications. It contains the Java Cryptography Architecture (JCA)[28], that provides a framework for common cryptographic mechanisms (e.g. encryption, key generation, key agreement) that are convenient, for example, the establishment of the secure channel between the user and the RSIGN Backend. As an extension to this architecture, Bouncy Castle[29] was used as an implementation of the security mechanisms used in the establishment of this channel. Specifically, the DHKE implementation allows the generation of key pairs and later the agreement of a shared secret key between the user’s endpoint and the RSIGN Backend. The use of this
proprietary extension guarantees a safer way to use these mechanisms because they are tested and are also widely used in security solutions.

To make RSIGN a web service, Spring Boot was used. This technology contains an embedded Tomcat web server connected to a local port. The client application that wants to interact with the RSIGN as a web service, uses a URL that contains the intended REST API method. Both entities exchange data using POST requests as custom Java objects, particular for this interaction. For example, the request of a signature creation includes: the PIN and the hash values of the data that is meant to be remotely signed and the PIN for signature authorisation. The request is made using a URL of type: \texttt{https://address:port/sign}.

Regarding the database, Hibernate, a Java framework for database interaction was used. PostgresSQL was the database endpoint used. It requires a user, in this case the RSIGN Backend, to be authenticated towards it to be able to query for data.

### 5.3 Communication Protocol

To accomplish the interaction between the entities that compose the RSIGN solution, a communication protocol was designed. In this solution, requests made by the user include data that is intended to be accessed by different components in the remote service. For instance, the signing request, build by the user endpoint, includes the user name and an enclosing object containing the encrypted hashed data (to be signed) and the PIN. The user name is relevant for the Client Portal to identify him/her in the system, but the hashed data is intended to be received by the RSIGN Backend and the HSM only.

This protocol defines java objects that contain the application’s data, where objects may contain other objects. Figure 5.2 displays the objects and how they are related to each other.

The type of messages can be divided into three categories. The Client Portal related messages, at the higher level, that contain information such as username and password so that the user can be identified in the system. The Qualified Message is related to the secure channel that exists between the user and the RSIGN Backend, it contains the encrypted message and information to check its validity. Then there is the messages that contain the application data necessary to carry out core operations, e.g. the hashed data to be signed and the PIN.
5.4 Storing signing keys

As stated in Chapter 4, keys are stored in a Multicert internal database. Databases allow persistent storage of information that can be retrieved in an efficient manner. The database domain is organised as displayed in Figure 5.3.

To have a more scalable database and to unambiguously finding wrapped signing keys, queries are made using a different table containing the certificate information (for the corresponding public key).

On signing key pair generation, users’ wrapped signing keys and the corresponding public key (both in Base 64 encoding) are stored using table \textit{WRAP.INFO}. The mentioned public key is used as an index to the associated wrap. This is needed because, at this moment, no certificate under this public key exists. As mentioned before, the Backend must hold for an
activation notice stating that certificate generation was successful. Once that notice is received, the certificate information is given and it is then stored and associated with the existing entry in the \textit{WRAP}.\textunderscore\textit{INFO} table. The information that is given to RSIGN is:

- Certificate serial number;
- Certificate domain name;
- Issuer Serial number;
- Issuer domain name.

The Certificate Serial number and the Issuer Domain name make a certificate unique, in legal terms. The Certificate Domain name and the Issuer Serial number (of the issuer’s certificate that allows verification of the given certificate) are extra information that may be used by back-office services. The table \textit{CERTIFICATE}.\textunderscore\textit{INFO} and the \textit{ISSUER}.\textunderscore\textit{INFO} separates the issuer certificate information from the users’. This decoupling, optimises database insertions in the sense that the same CA may issue several certificate for users’ public keys thus avoiding having to also insert the issuer information for each new user certificate.

When the signing key is needed, i.e. on signing and changing the PIN operations, the request to RSIGN made by the Client Portal includes the certificate information, i.e. these four fields, associated with that signing key. The RSIGN Backend queries the database for the wrapped signing key. Then the rest of the operations are carried out with the help of the HSM.

5.5 Interacting with the HSM

RSIGN relies on an HSM for securely perform cryptographic operations. A device such as a HSM is required to create remote qualified signatures[2]. Allowing the RSIGN Backend to communicate with a HSM requires the use of an interface for this interaction. HSMs usually provide a specific and proprietary API, but to make solutions less dependent of specific HSMs, they usually offer the means to interact with these devices using a PKCS#11 interface.

Figure 5.4\textsuperscript{1} displays which modules are layered to accomplish the interaction between the RSIGN Backend and the HSM. The web server (Application) contains an object that represents the HSM, as if it were local. This object is a module of a Multicert internal product called
HSM-Interface. It is an abstraction of different HSM implementations. This product did not have a proper PKCS#11 implementation, so it was extended to include it, relying on the IAIK PKCS#11 wrapper[26] responsible for translating commands made in Java, to lower-level calls made to the HSM.

![Figure 5.4: Different layers involved in the communication with RSIGN’s HSM](image)

As defined in the PKCS#11 document[12], the RSIGN Backend must provide the location of the PKCS#11 module (HSM Module), provided by the HSM manufacturer, to connect to the HSM. Every time the RSIGN Backend needs to interact with the HSM, it must open a session. It does so by providing the PIN of the slot it is using (defined during the HSM installation). The commands the RSIGN Backend makes to the HSM are to:

1. Generate a symmetric key (for master key);
2. Generate an asymmetric key pair (for signing);
3. Generate a Random (for PIN generation);
4. Derive a key (for wrapping key generation);
5. Wrap a key (for wrapping signing keys);
6. Unwrap a key (for unwrapping signing keys);
7. Create a signature (for user provided data).

The PKCS#11 defines these commands and their parameters. Some of these operations parameters (1, 2, 4, 6), require that a key template is given for the key it is generating, deriving or unwrapping. The key template defines the attributes that each key possesses is described in Table 5.1.

---

1 In grey are the modules that make the HSM as a local object, in white is the module that is provided by the HSM manufacturer.
### CHAPTER 5. RSIGN’S IMPLEMENTATION

<table>
<thead>
<tr>
<th>Key type</th>
<th>Obtained in operation</th>
<th>Relevant attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master key</td>
<td>Generate symmetric key</td>
<td>key type: CKO_SECRET_KEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>key algorithm: AES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>token(^2): TRUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>derive: TRUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>extractable(^3): FALSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value len(^4): 32</td>
</tr>
<tr>
<td>Private signing key</td>
<td>Generate asymmetric key pair</td>
<td>key type: CKO_PRIVATE_KEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>key algorithm: RSA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>token: FALSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sign: TRUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>extractable: TRUE</td>
</tr>
<tr>
<td>Public signing key</td>
<td>Generate asymmetric key pair</td>
<td>key type: CKO_PUBLIC_KEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>key algorithm: RSA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>token: FALSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>verify: TRUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>modulus bits(^5): 2048</td>
</tr>
<tr>
<td>Wrapping key</td>
<td>Derive key</td>
<td>key type: CKO_SECRET_KEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>key algorithm: AES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>token: FALSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wrap: TRUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value len: 32</td>
</tr>
<tr>
<td>Unwrapping key</td>
<td>Derive key</td>
<td>key type: CKO_SECRET_KEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>key algorithm: AES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>token: FALSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unwrap: TRUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value len: 32</td>
</tr>
</tbody>
</table>

Table 5.1: PKCS#11 attributes\(^{33}\) for keys of RSIGN.

The attributes for each key template that are not displayed in the table follow a strict policy, if they are not needed to be true, they are surely false (e.g. the master key is not needed to wrap other keys, hence the wrap attribute for the master key template is set to false).

In regard to key sizes and algorithms, keys in RSIGN have the values recommended by the National Institute of Standards and Technology (NIST)\(^{34}\). The size of the keys also considers the performance of the operations. If key sizes were longer than necessary, the latency for each operation would be hindered. Regardless, RSIGN implementation is flexible in the sense that it can be easily changed to perform its normal operation with different key sizes.

\(^2\)true if the key is persistent in the HSM (token object). False otherwise (session object).

\(^3\)size of the key in bytes.

\(^4\)true if the key can be extracted, including when wrapped. False otherwise.

\(^5\)size of the key in bits.
PKCS#11 functions may or may not receive key templates as input. However, functions must always receive a mechanism as input. Mechanisms allow for the HSM to know which algorithms to use in the function. The most relevant mechanism, in RSIGN, is the one used for deriving the wrapping key, using the master key as the base key for derivation: \texttt{CKM\_AES\_CBC\_ENCRYPT\_DATA}. What is particular about this mechanism is that it can receive data as a parameter, in this implementation, a SHA-256 hash of the PIN. Other mechanisms were chosen based on the type of keys they receive as input (e.g. the mechanism \texttt{CKM\_RSA\_PKCS\_KEY\_PAIR\_GEN} is used to generate the RSA signing key pair).

### 5.6 Client application: myRSIGN

Even though the core of the solution for creating remote qualified signatures is in the server, since it is a product to be used by non-technical users, a Graphical User Interface (GUI) was implemented. The choice of a mobile application was motivated by the wide use of smartphones and the convenience it offers for users to be mobile while being able to do online operations.

This section presents myRSIGN (my Remote SIGNature). The ultimate goal of this application is allowing users to obtain qualified signatures for local files he/she wishes to. To be able to do this, the operations that the user has at his/her disposal are:

**Signup**  This operation allows users to be registered in the Multicert’s Client Portal. The user must give as input its email and password. If the user is already registered under the email that was provided, it is notified of that error, as shown in Figure 5.5. If the signup is successful, the user is redirected to the login layout.

![Figure 5.5: layout for the operation, showing that the given username is already taken.](image-url)
Login To login, the user must use the credentials that were previously registered in signup. The request to login also includes the data necessary for establishing the secure channel with RSIGN Backend, as described in Section 4.5. On success, i.e. credentials verified and secure channel established, the user is redirected to the main menu, which has the layout depicted in Figure 5.6.

![Figure 5.6: after successful login](image)

Register in RSIGN After login, to be registered in RSIGN, the application sends the request through the secure channel. The operation is complete when the user is able to see its PIN. This PIN will show for a limited time and only when the user clicks 'show PIN'. This is to avoid a shoulder surfing attack. After this operation, the user cannot retrieve the PIN because the application does not store it. He/She must memorise it. The displaying of the PIN after successful registration in RSIGN is illustrated in Figure 5.7.

![Figure 5.7: Layout that can display the PIN.](image)

Change PIN This operation allows users to change the PIN, as a convenience. Of course, the user must choose a value of a defined minimum size.
5.7. SUMMARY

Signing This operation, the most important one, allows the user to choose one or more files to be remotely signed. To do this, the user must input its correct PIN. The screen for choosing the file(s) is shown in Figure 5.8. Note that the hashing of the file is done on the client side, the server never has access to the file(s) to be signed.

![Figure 5.8: Screen to choose a file to be signed](image)

After the file(s) are signed remotely, the user can verify the signature value, as displayed in Figure 5.9.

![Figure 5.9: screen showing the signature result of a given file.](image)

Error handling At any given operation, if an error occurs in the remote service, the user is notified with sufficient error information. For example, invalid credentials on login appear as displayed in Figure 5.10.

5.7 Summary

In chapter Chapter 4 RSIGN was presented. This chapter details the implementation that resulted in a complete product that creates Remote Qualified Digital Signatures.
Being an enterprise solution, RSIGN was implemented using several tools. These include: database interaction (PostgreSQL and Hibernate) and web services (Spring Boot) frameworks for Java (the main implementation programming language for this solution).

The service is made available by a REST interface. A message protocol was created to exchange messages. This protocol consists of sending well-defined messages to exchange data in a way that is flexible and secure. This was motivated by the need for different components to obtain a subset of the data in each message exchanged.

The signing keys are stored in a database outside RSIGN. They are unambiguously identified by the certificate information that was issued for a given signing key pair. One user registration, certificates are generated outside of RSIGN Backend’s scope. For that reason, this Backend waits for an activation notice stating that the certificate generation was successful and that therefore the registration is complete. Only then the signing key, which is stored in the Key database, is eligible to be used for signing operations. This is due to the asynchronous way the registration in RSIGN is processed.

The interaction with the HSM was done following the PKCS#11 standard. To abstract the HSM for this solution, a component from Multicert called HSM-Interface was extended to allow interaction through PKCS#11. This was done using a wrapper for PKCS#11 that translates more generic commands into PKCS#11 HSM PKCS#11 modules. The type of keys used in RSIGN are defined through a template containing the necessary attributes. Besides the key algorithm and the key length, attributes such as if the key is persistently stored in the HSM are important to implement the designed solution.

RSIGN goal is to be a service that is accessible to users through their mobile device. A
mobile application in Android was developed to allow regular non-technical users to use the system. The application is of simple design and in just a few steps, the user may obtain qualified signatures for documents it chooses.
This chapter states a qualitative and quantitative analysis of RSIGN, considering the design and implementation described in Chapter 4 and in Chapter 5.

A comparative analysis (Section 6.1) between RSIGN and the existing solutions (previously mentioned in Section 3.3.5) is stated. The bibliography on the existing solutions is limited regarding whole of the features described for RSIGN. For that reason this comparison is limited to comparing the features of RSIGN with some of the features of the existing solutions.

Following STRIDE[35], a security analysis (Section 6.2) states how the solution overcomes or mitigates different type of threats. This section considers the Section 4.1 to discard some of the obvious threats mentioned in Section 4.1, such as the user giving away his/hers credentials.

Since RSIGN is intended to be a commercial product, the efficiency of the solution, specially in terms of latency, is an important property of the system so that users can use it in a convenient way. A quantitative analysis is made to the Implementation described in Chapter 5.

This chapter concludes with some final considerations and recommendations (Section 6.4) for the previously mentioned implementation.

6.1 Comparing RSIGN with Existing Solutions

The existing solutions cannot be extensively compared to RSIGN in the same way. This is due to the fact that some solutions do not fully describe all the features required for qualified signatures’ creation, e.g. signing authorisation made in a secure way but not signing key protection or vice-versa. This section will compare some of those solutions’ features with RSIGN’s.

Regarding the protocol for protecting signing keys, the solutions presented by Orthacker et al.[21] and Rath et al.[16] mention that signing keys are not stored inside the HSM because of storage limitations, instead they are protected to be stored externally, just like RSIGN. The
main difference between this solution and the RSIGN solution is that the first does not protect a given user’s signing key with the PIN inside the HSM (recall Equation 3.2). It rather outputs the wrapped signing key and then uses a user public key to encrypt the key wrap. The private key that allows the decryption of that object is wrapped using a PIN derived symmetric key. This decoupling between the user PIN usage and the protection of the signing key requires that the service must store, for each user, two separate signature data objects. In RSIGN this is optimised because the PIN is used directly in the protection of the signing key, by using it as a parameter to the generation of the wrapping key (recall Equation 4.1) resulting in a single object that contains the signing key.

Rath et al.[16] do not detail how the signing key is protected therefore it cannot be compared to the way RSIGN does it.

The solution proposed by Zefferer et al.[17] relies on external Austrian modules for qualified signature creation, no information is provided on how these modules protect the signing keys, besides that it relies on a HSM and that a PIN is required for signature authorisation.

Being the only client-side system of the ones presented, Certification on Demand[14] suffers from the hardware dependency that is undesired for users to carry out online operations. As emphasised before, RSIGN is a solution in which the user may obtain qualified signatures for the data him/her intends by having the SSCD on a remote service. For this reason, the users’ mobile device computation is reduced, instead heavy computation is left for remote components.

However, there is the case where RSIGN might have a disadvantage when compared with other solutions, specially [21]. When the user inputs an incorrect PIN, the error is only detected on unwrapping the signing key inside the HSM, after the wrap is identified and retrieved from the database. There are a few steps that lead to requesting the user to input the correct PIN, after a wrongful try. This is the cost of not storing the PIN in the system.

In terms of the communication between the user and the service, existing solutions rely on components which are not stated to be secure, such as web servers. RSIGN includes a secure channel between the RSIGN Backend and the client’s application, which besides protecting sensitive data, does not depend on any third-party entity, e.g. mobile network. This is an advantage because there is no commercial lock-in in the implementation of RSIGN, i.e. RSIGN relies only on the infrastructure it is meant to be deployed in: Multicert’s.
6.2 Security Evaluation

To qualitatively evaluate RSIGN in terms of security, a threat model must be defined in order to understand how the system responds. STRIDE\cite{35} was used to identify the threats to the solution.

Spoofing identity

1. An attacker may try to brute-force his/her way into obtaining a given user’s email and password, thus acting as an authorised user;

2. An attacker may try to brute-force his/her way into obtaining a given user’s PIN, and attempt to authorise a signing operation;

3. An attacker may obtain the credentials and the PIN of a given user;

4. A fake Client Portal may pose has an authorised one in order to obtain the user’s credentials and PIN.

Tampering with data

5. An attacker may tamper with the commands made through the REST API;

6. An attacker may tamper with the file(s) being signed for an authorised user replacing it/them with a file(s) of is own;

7. An attacker may tamper with messages being sent in the secure channel between the user and the RSIGN Backend;

Repudiation

8. An authorised user may deny having authorised signature creation for a document it chose;

Information Disclosure

9. An attacker may eavesdrop on a signing request in order to obtain the PIN or the file being signed;
10. An attacker may gain access to the Key Database and try to obtain a given user’s signing key;

11. An attacker may gain access to a file in the signing request to obtain its contents in plain text.

**Denial of Service**

12. An authorised user may input wrong PIN, causing RSIGN failure;

13. The MTC Backend may retain a user request, preventing it from reaching RSIGN.

**Elevation of Privilege**

14. The Client Portal may try to change the user that is requesting the signature operation;

15. A malicious system administrator may try to obtain the signing key and make a signing request to RSIGN for a file it chooses;

Given the threats identified above, the way RSIGN solves or mitigates these problems is stated. Note that the modelling of the threats to RSIGN takes into consideration the trust model defined in Section 4.1.

In terms of **Spoofing identity** threats, they consist of an authorised entity posing as an authorised one and focuses on the use of RSIGN with users already registered.

In the case where the attacker gains access to a given user credentials, this means brute-forcing both the email and password. This is avoided, in the Client Portal, by limiting the attempts a given user can make to login in the system, requiring a period of time before a new attempt is made. Even in the case it does gains access to the correct user credentials, the attacker still needs to know the user’s PIN in order to perform serious hindrance to the user and to the system (signing or changing a PIN).

Since the PIN is only verified on the unwrapping operation, performed by the HSM (the PIN is only known to the user), brute-forcing the PIN is also avoided by limiting the attempts a user can make. When this occurs, the same tactic as with user credentials is used, but a longer period of time before the user can attempt the PIN again is required. This increase is due to the catastrophic consequences of an attacker finding a given user’s PIN.
Nevertheless, there is the case where the attacker finds a given user’s email, password and PIN. This is the case where the trust model mentions that the user is the only one that knows the email, password, and PIN. RSIGN depends on the undisclosed access of this information.

The last threat identified for this type of threats is a fake Client Portal posing as the Multicert’s Client Portal in order to obtain credentials and PINs from users. This is avoided by having the Client Portal authenticated towards the client application through an HTTPS connection. The client application has the Portal public key certificate which allows a secure server-authenticated connection. For this, the certificate’s validity must be verified regularly.

Regarding threats that Tamper with the data, it is related with maliciously changing data being exchanged between authorised entities in the system. This concerns the path that messages go through between the client application and the Client Portal.

For instance, an attacker may succeed in changing a message sent by the user endpoint and meant to be received by the Client Portal. However, since a Secure Socket Layer (SSL) connection exists between the two, data authenticity mechanism are in place and prevent such attack.

Changing sensitive user data, such as a file to be signed in a signing request, is also avoided by having the secure channel on top of existing TLS’s which doubles the integrity mechanisms. Particularly, changes in request going through the secure channel does not go unnoticed because of the HMAC value that authenticates the data being sent. This also stands for the IV which is sent in plain text, but authenticated through the hash. At most, an attacker, by tampering with a message, performs a Denial of Service attack (tampered messages are rejected on arrival).

As for Repudiation, the expected case related to digital signatures, is the user deny having authorised signature creation. This could be supported by the fact that signatures are actually being computed remotely but, as stated before, RSIGN avoids this by having the users’ signing key protected using their PINs, which the system has no knowledge of, since it is not persistently stored. Given proper verification of the signing key usage and creation by third-party entities, the user would have no legitimacy to state that it did not authorise remote qualified signature creation, since only him/her could do it. Once again, this is also guaranteed by the robustness of the components in the Qualified area, mentioned in the Section 4.1.

Information Disclosure deals with the privacy of the data, i.e. undisclosed for unau-
thorised entities. In RSIGN, examples of related attacks include an attacker eavesdropping a signing or change PIN request to try to obtain the PIN that authorises signature creation. This is avoided by having the two-layered secure channel between the client application and RSIGN encrypting such sensitive data.

In the case where the Key Database or the MTC Backend is compromised and the attacker obtains the signing data and as stated before, the signing key cannot be undisclosed outside the HSM, because the key that is used to obtained it, the master key, is only known by the HSM itself.

Files being signed are never sent to the service. Instead, the hash of the files are sent, preventing even the service from gaining undisclosed access to their content. This is motivated by the fact that, to create a qualified signature, the service does not need the file itself. It only needs its hash value, so it can use the file owner’s private signing key to encrypt it.

**Denial of Service attacks** aims at making services unavailable for users by disrupting them. Such attacks are usually accomplished by flooding the service with frequent request that create a heavy load on the service.

The threats to RSIGN are related to the attacker making various requests to the Client Portal. At user registration in RSIGN there are no mechanisms that avoid this kind of attack. This is because there needs to be, to a certain extent, trust that the registration is done in the expected way, i.e. by non-malicious users that can be properly verified. As stated before, RSIGN is part of a service provided by Multicert. This makes RSIGN a valid product for users to have remote signature data that they can (indirectly) use. The expected degree of trust is accomplished because of the security measures applied to online purchases. Regardless and as mentioned before, login, signing and changing the PIN are operations that avoid an abnormal request rate by limiting the attempts of authentication and authorisation using the credentials and the PIN, respectively.

Denial of Service threats are related to the Availability of the systems. As stated before, this architectural requirement was not extensively considered, thus allowing stronger Security and Performance tactics. Nonetheless, Availability is considered in the existing Multicert’s infrastructure making it a secure barrier between RSIGN and the outside world.

Regarding **Elevation of Privilege** threats, it considers attacks that are performed from
components of the system. As stated before, intermediate components between the client application and the Qualified area are not fully trusted. For this reason, a secure channel between the trustworthy components, i.e. the user endpoint and the components in the Qualified area, is established before any sensitive data is exchanged. Because of this, even if the Client Portal attempts to change the user that is making the request for signature creation, for instance one it knows the credentials and PIN, it still will not be able to succeed in the attack because the PIN in the request that authorises signature creation is protected under the properties of Confidentiality, Data Authenticity and Freshness. This means that the Client Portal cannot know the user’s PIN because it is encrypted, it cannot change the data in the message without going unnoticed because of the HMAC and cannot store the message for later usage, because the user endpoint can verify if the message is recent.

6.3 Performance Evaluation

RSIGN allows users to securely obtain qualified signatures that are created remotely. Security is clearly an important requirement, specially because of compliance with eIDAS[2]. Regardless, in order to have a usable solution where users can rapidly perform the operations, performance is also important (poor performance leads to poor usability). Performance is, then, a significant requirement taken into account in the design of the solution. This section evaluates the performance requirement based on the implementation described in Chapter 5.

To test the solution, the machine that is running the service has an Intel(R) Core(TM) i5-3340M CPU running at 2.70 GHz, with 8 GB of memory, and the operating system Windows 8.1 Pro. The HSM is the Crypto Server simulator[36] and both the Client Portal and the RSIGN Backend were running in separate processes. Client-side an android device version 4.4.2, physically close the service’s machine, running at 1.7 GHz in a Octa-Core ARM Cortex-A7 with 2 GB of memory was used as the platform to run myRSIGN. Both devices were connected to the same internal network, at Multicert.

Regarding the profiling tools that were used for measuring execution times, for the RSIGN processes it was JProfiler[37] and the Android Studio built-in Monitor[38] for myRSIGN.
6.3.1 Overall Performance

In order to understand the overall performance of the solution, Figure 6.1 displays the average time for each operation. Ten tests were run for each operation with both myRSIGN and RSIGN in a steady state. Furthermore, it was assumed that the user had already successfully concluded the signup operation.

![Figure 6.1: Latency for each RSIGN operation, measured in seconds (standard deviation included).](image)

All four operations required a communication between the client application, the Client Portal and the RSIGN Backend. As can be observed, the Register operation is the one that takes longer: 2,752 seconds. This is due to the cryptographic computation involved in this process. As described before it is in this operation that a signing key pair and the PIN are generated for the user, the signing key pair generation being the most computationally demanding on the service. After the private signing key is protected and exported, the certificate is generated and stored. For this operation several accesses to the HSM and to the databases are made, which explains the operation duration.

The second longest operation is the login (2,752 seconds). It is in this operation that the secure channel between myRSIGN and the RSIGN Backend is established which, as described in Section 4.5, involves cryptographic computation, performed both by the client application and the RSIGN Backend.
6.3. PERFORMANCE EVALUATION

For the signing operation, a file of 1 MB was chosen to be signed. This is the most common (and most important for that matter) of all the ones tested. It is also the one that has the lowest latency. The reason why changing the PIN has higher latency than the signing operation (one would assume changing the PIN is computationally easier) is because it involves the unwrapping and wrapping of the user’s signing key whereas the signing only involves the unwrapping of the signing key.

6.3.2 Signature Creation

As mentioned before, in Section 4.6.4, the user can send more than one file in the signing request. Figure 6.2 illustrates the test of increasingly send more files in the request. All the files that were signed were of 1MB of size.

![Figure 6.2: Latency of the signing request, depending on the number of files to be signed.](image)

As can be observed, as the number of files in the requests increase so does the time it takes in the operation. For instance, for one hundred files the latency is 19,603 seconds. This operation requires that the hash values of the one hundred files is calculated added to the request and sent through the secure channel along with the authorisation PIN. On receiving the request, the service retrieves the wrapped signing key in the Key database and inside RSIGN the signing key is (with the help of the PIN) obtained in the HSM which is then used to sign the received hash values. The result is sent to the client application through the secure channel.
Comparing to the alternative of only including a file per request, the advantage of the implemented approach clearly requires less interaction by the user with myRSIGN (inserting PIN for each file) and between the components in the system. Note that the wrapped signing key would have to be retrieved from the database and unwrapped every time a request, containing the hash value of a single file is made.

6.4 Final Considerations

This section states what needs to be considered regarding the solution and why the product this document presents can be deployed and used by users who wish to obtain qualified signatures.

One way to qualitatively evaluate the solution is to verify if the requirements, defined in Chapter 1, were fulfilled.

Requirement 1 states that the solution must be secure and reliable, in the way it provides the signing functionality. This is accomplished by managing the users’ signing keys in a way that only trusted components may use them, with user authorisation, to create qualified signatures. Furthermore, the exchange of sensitive data is done through a secure channel that provides Confidentiality, Data Authenticity and Freshness to sensitive data (this overlaps with Requirement 3).

Regarding Requirement 2 where the user interface must allow to be used in the users’ mobile platform. This is accomplished by making the service available through a REST API. The invocation of this interface is implemented in a mobile Android application called myRSIGN. The only major issue in the implementation of the solution in different mobile platforms is that they should allow for the hash of the data, to be remotely signed, to be calculated in the client-side. If not, it is not that the data is undisclosed to untrusted components, because it is sent through the secure channel established with the Qualified area, but creates the overhead of sending the whole data plus the Qualified area has access to it in plain text.

Requirement 4 states the usage of a remote HSM in which qualified signature creation takes place. This requirement is related to compliance with eIDAS (Requirement 5) because the latter requires the use of a Secure Signature Creation Device (SSCD). HSMs are robust and secure
devices and because of the flexible interface it supports, PKCS#11, it is not limited to the same HSM.

About Requirement 5, the explicit authorisation of the user is done through the use of a PIN that, since it is only known to the user, it is the only way to obtain his/her signing key in the HSM. The signing key only exists in the HSM while qualified signatures are being created or when, for a brief moment, it is unwrapped and wrapped again with a key derived using a new PIN (Change PIN operation). Even though the PIN is protected from intermediate attackers, its usage does not represent the strongest authorisation mechanism. For that reason alternatives ways should be considered.

Another way to qualitatively evaluate the solution is through its integrability in the Multicert Infrastructure. The solution implementation (Chapter 5) mentions how RSIGN can be applied to the existing components (MTC Backend and Client Portal). Since RSIGN is a separate web server, its deployment could involve making the REST calls from the MTC Backend, and connecting the Key database and HSM. Furthermore, even though the implementation used a given configuration in terms of the keys (Section 5.5), it can work all the same by changing the configurations in specific configuration files. This can, for example, allow for the use of a different master key size.

In terms of performance, it is important to understand that the tests where run locally. A network overhead would certainly mean higher operations latency. What these tests aimed at was understand how much each operation performs in the use of the HSM. In that sense, the latency of each operation is low considering that they require complex security mechanisms to be applied. Since the ultimate goal of this solution is the signing operation, being the one that takes less to be computed and is the one that is probable to be requested more times, this is an advantage. Login and Register are the operations that have the highest latency, which does not mean an excessive amount of time for each operation. Regardless, they are both operations that do not occur repeatedly. In each interaction with the service, the Login only happens once and the Register operation only happens once for all interactions with RSIGN, for a given user. Changing the PIN is a operation that is expected to be seldom requested. Nonetheless, it has a low latency.

As mentioned in Section 6.1, there is a disadvantage in not storing the PIN in the service, i.e. when a wrong PIN is given, it is only detected in the unwrapping of the signing key, after some
demanding steps have already occurred. In terms of performance, this means a 1,132 ± 0,353 seconds between sending the request with a wrong PIN and receiving the response stating that the PIN is wrong. Given the time that it requires, in terms of performance, the identified disadvantage has no great impact in the performance of the operation.

Regarding the number of files in the signing requests, RSIGN performs better with the increase of the number of files, i.e., it does not pose a great impact on the operation latency considering that users usually do not wish to sign large number of files in the same request (e.g., one hundred files in one request is not the most expected use case).

**Summary**

This chapter provides a evaluation of the solution described in Chapter 4 and in Chapter 5.

A comparative analysis between the existing solutions, described in Chapter 3, is stated. RSIGN is outlined from the other solution due to the convenience of no extra hardware on the client-side and the secure way keys are managed outside the HSM.

In terms of security, the solution is analysed considered the threats to it. RSIGN mitigates most of the problems by having the communication protected in a way that the data is only accessed strictly by entities that might use them.

Regarding the performance of the solution, tested locally, is not heavily hindered by the extra cryptographic tasks in the HSM, the RSIGN Backend, and the user endpoint. Furthermore, a large number of files in the signing request does not pose an unexpected high latency.

This chapter concludes with a global appreciation of the solution presented, by checking the fulfilment of the requirements proposed in Chapter 1. RSIGN performs Remote Qualified Signatures as it was set out to do.
Motivated by the increasing convenience of mobile platforms, users perform their daily online activities using those devices. The interaction between services, e.g. e-Commerce, and users gives the opportunities for malicious entities to obtain and/or tamper with the data being exchanged. Strong security mechanisms must be used to allow for users to carry out even sensitive operations.

One security mechanism that provides Authenticity and Non-Repudiation to data are Qualified Digital Signatures[2]. These special kind of signatures are more robust than regular ones because they are created in a secure environment, i.e. inside a SSCD, and can be reliably verified. When receiving a message along with a qualified signature, the service can trust that the data and the user that sent it are authentic and the user cannot deny to have authorised the signature creation, since only he/she could.

Existing work on the topic of qualified signatures discusses the improvements of Directive 1999/93/EC[1] (where Qualified Signatures were first introduced) in the most recent Regulation 910/2014, commonly known as eIDAS[2], regarding the possibility of qualified signatures being created remotely to the user. This improves usability because users do not need to carry an SSCD in order to create qualified signatures. Instead, the device is placed at a remote service.

By having the qualified signatures created remotely, the user, owner of the signing data, must trust that not even the service that creates qualified signatures in his/her name cannot perform unauthorised signing operations. Furthermore, the service must be sure that the user really authorised signature creation, otherwise Non-Repudiation is not assured.

Some existing solutions allow for remote qualified signature creation. However they require extra hardware due to the strict requirements of user authentication. This means that the user’s endpoint is not platform independent and that the service relies on the actions of a third-party, i.e. the network carrier.
To promote a more convenient way for user to obtain remote qualified signature creation, and to overcome the problems identified for the existing related work, RSIGN is proposed.

Remote SIGNature (RSIGN) is an HSM-based solution that manages users’ signing keys and allows for sensitive data to be exchanged through a secure channel. Due to storage limitations of HSMs, keys are stored outside its environment. To assure that only the HSM with the authorisation of a given user can create the qualified signature with the corresponding signing key, the latter is protected using both user and HSM private data.

The solution herein proposed intends to be deployed in the existing Multicert’s Infrastructure[22]. Particularly it allows for alternative ways to obtain signatures and to verify them. Furthermore, user authentication is assured by already deployed Multicert components.

To have a functional product, a client Android application, named myRSIGN, was developed to serve as the UI for users. This mobile application allows for users to chose local file(s) to be remotely signed by RSIGN and obtain the corresponding qualified signature.

7.1 Conclusions

RSIGN overcomes the nuisance of users’ having a local extra hardware in order to obtain qualified signature for data they choose. Furthermore, compared to other solutions, it does not require any other device besides the users’ mobile device. Since the service is deployed in the same infrastructure, Multicert’s, there is also no third-party dependency. When compared with a similar solutions, it also optimises storage of signing keys outside the HSM.

A security analysis is further supported by the STRIDE[35] model, were possible threats were analysed. Some of the threats are relaxed (though not ignored) because of some assumptions that were made. The trust model allowed for a deeper development of the core aspects of the solution which rely mostly on signing key management and sensitive data exchange. Security of the endpoints was the major assumption included in the trust model.

Regarding performance, the solution has a low latency, specially on the most important operation: signing. This advantage also holds for request where the number of files increases.

In terms of fulfilment of the requirements and goals set for this thesis, the implemented solution successfully tackled the problem of Remote Qualified Digital Signatures resulting in a
7.2. Future Work

In this section, what comes next for RSIGN is stated. It also mentions some improvements that must be taken in account.

RSIGN is meant to be a deployable service in the existing Multicert Infrastructure. This will allow for real users to make use of this service. Most of the services are available through web applications. Due to the requirement of client-side hashing, this solution should interface with users through a mobile application. This would mean improve the user experience of the existing myRSIGN. For example, the layouts are basic and lack functionality regarding the choice of files to be signed. The presentation of the signature value after a successful signing response must also be improved.

The use of a PIN as the authorisation code for qualified signature creation is not ideal, since the user needs to memorise it and because it is usually a four digits number, which makes it susceptible to shoulder surfing and brute-force attacks. The latter could be mitigated by transforming the PIN into an One Time Password (OTP). This would require that the service reverses this OTP in order to obtain the PIN and use it to compute the wrapping key.

Regarding the identified relaxation of Availability mechanisms, as mentioned before, the existing Multicert components comprise such mechanisms. However RSIGN itself must be deployed in a strict environment due to the existence of a Qualified area. The latter should be in an internal network with strict access from authenticated components.
Bibliography


[18] CMKC Cuijpers and Jessica Schroers. eidas as guideline for the development of a pan european eid framework in futureid. 2014.


7.2. FUTURE WORK


