UNIVERSITÀ DELLA CALABRIA

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Master Degree Course in Computer Science

Master's Thesis

Automation of digital signature workflows in the italian public administration

Supervisor:
Prof. Giovambattista Ianni

Candidate:
Alessio Scarfone
Matr. 187178

ACADEMIC YEAR 2017/2018

Abstract

This thesis work aims to automate and simplify the use of digital signature in the public administration. The **Digital Signature** is the computer equivalent of a traditional autograph signature written on a paper document and guarantees **authenticity**, **integrity** and **legal validity**. Although commonly used in the italian public administration, the "**Qualified Electronic Signature**", has some drawbacks. In particular, the qualified electronic signature needs a special device for its creation, it presents poor integration with other software and implies long and repetitive usage processes. Currently, creation, management and verification of digital signatures are conditioned by the use of software specially developed by the suppliers of devices or digital certificates (eg. ArubaSign).

This work led to the development of two software modules: **a command line tool** that allows the signing of documents, using a qualified digital signature token that follows the PKCS #11 standard, and an **extension for the Chrome browser** that integrates these features within a browser. This allows to sign documents directly from a web browser without having to use another tool and it makes using digital signature faster and easier for users. The documents signed with these tools comply with the EU Regulation n.910/2014 on electronic identification and trust services for electronic transactions in the internal market (***eIDAS Regulation***) and supports *CAdES* and *PAdES* standard formats.

The developed software can be easily extended to support other signature formats or add additional features, such as verifying the validity of signatures, or direct interaction with other public administration systems that require digitally signed documents as input.

Sommario

Questo lavoro di tesi ha lo scopo di automatizzare e semplificare l'utilizzo della firma digitale nella pubblica amministrazione. La **Firma Digitale** è l'equivalente informatico di una tradizionale firma autografa apposta su un documento cartaceo e garantisce **autenticità**, **integrità** e **validità legale**.
In particolare la "**Firma Elettronica Qualificata**", comunemente utilizzato nella pubblica amministrazione italiana, presenta alcuni inconvenienti. La firma elettronica qualificata richiede l' utilizzo di un dispositivo apposito per la sua creazione, presenta scarsa integrazione con altri software e implica processi di utilizzo lunghi e ripetitivi. Attualmente, creazione, gestione e verifica delle firme digitali sono condizionate dall'utilizzo di software appositamente sviluppati dai fornitori dei dispositivi o dei certificati digitali (es. ArubaSign).

Il lavoro ha portato allo sviluppo di due software: **un tool da linea di comando** che permette la firma di documenti, utilizzando un token per firma digitale qualificata che segue lo standard PKCS #11, e un **estensione per il browser Chrome** che integra queste funzionalità all'interno di un browser. Questa permette di firmare documenti direttamente dal browser senza dover utilizzare un altro strumento apposito e ciò rende l'utilizzo più rapido e semplice per gli utenti. I documenti firmati con questi strumenti sono conformi al Regolamento UE n.910/2014 in materia di identificazione elettronica e servizi fiduciari per le transazioni elettroniche nel mercato interno (***Regolamento eIDAS***) e supportano i formati standard *CAdES* e *PAdES*.

I software sviluppati sono facilmente estendibili per supportare altri formati di firma o aggiungere funzionalità ulteriori, come può essere ad esempio la verifica della validità delle firme, o l'interazione diretta con altri sistemi della pubblica amministrazione che richiedono come input documenti firmati digitalemente.

Contents

Chapter 1

Introduction

1.1 The context

With the adoption of the EU Regulation n. 910 of 23 July 2014 (2014/910) on electronic identification and trust services for electronic transactions in the internal market, known as **"eIDAS"** (electronic IDentification, Authentication and trust Services), a set of standards for electronic identification and trust services for electronic transactions in the European Single Market has been created. This regulation entered into force on 17 September 2014 and applies from 1 July 2016. All organizations delivering public digital services in an EU member state must recognize electronic identification from all EU member states. The regulation aims to overcome the obstacles of market fragmentation, lack of interoperability between nations and fighting against cybercrime. At this point, all public administrations must adapt to the new regulation and modernize their tools and methods.

Among these new standards that are becoming increasingly important, the Digital Signature tools play an important role. The *Digital Signature* allows to have a means of electronic identification and authentication recognized by all European countries, thus facilitating exchanges. The digital signature allows not only to sign a document but also to verify the identity of the signer, the origin of the document and to ensure that the information contained within it has not been altered in order to protect against fraud and counterfeiting. So, a digital signature, especially in its most secure form (called **Qualified Electronic Signatures**), acquires the same legal value as an autograph signature affixed to paper, and guarantees **data integrity**, **authenticity** and the **non-repudiation** of the signed document. A Qualified Electronic Signature is created by a **qualified electronic signature creation device** and it is based on a **qualified certificate for digital signature**.

1.2 Problems and goal

This thesis work focuses on the need to accelerate and simplify the use of digital signature. Currently, creation, management and verification of digital signatures are conditioned by the use of software specially developed by the suppliers of the devices, for signature creation, or digital certificates. These software are often slow, inconvenient to use and totally disconnected from other applications used daily to perform the normal administrative tasks.
The goal is therefore to create some more user-friendly tools to speed up the generation of digital signatures and their possible direct integration into other constantly used software, such as a web browser.

The development of this thesis requires the study of the various standards and the resolution of various problems:

- Interaction with digital signature creation devices.
- The signed documents must be valid and must comply with all standards.
- Integration of the signature creation capability in a browser using an extension.
- Communication between browser and signature creation device.
- Showing properly all the information needed to sign the document to the user.

1.3 Organization of the thesis

The thesis is divided into five chapters:

- In **Chapter 1 - Introduction**, the context, the objectives and the main problems faced in the implementation of the project are presented. It is also described a general overview of the structure of the thesis.
- In **Chapter 2 - Digital Signature Overview**, a general overview of the technology involved is described, starting from public key cryptography and arriving to the whole digital signature infrastructure, and a brief summary of its regulation that defines the standards and different formats is given.

- In **Chapter 3 - Java Command Line Tool for Digital Signature** it is presented a command line tool developed in Java, which allows the creation of a qualified electronic signature. It is described the general behaviour of the signature workflow, and it is reported a brief description of the external libraries used.

- In **Chapter 4 - Integration Digital Signature into Browser** it is presented the browser extension developed to integrate the digital signature directly into the browser. The extension uses the command line tool, previously developed, and provides a more user-friendly and more convenient method for creating signed documents.

- In **Chapter 5 - Conclusions**, the results obtained are analyzed and possible future developments together with problems that remained unresolved are identified and commented.

Chapter 2

Digital Signature Overview

2.1 Basic of Public-Key Cryptography

The Public-key cryptography, or *asymmetric cryptography*, is any cryptographic system that uses a pairs of keys: **public keys** which may be disseminated widely, and **private keys** which are known only to the owner. The two keys are different but related, one is used for encryption and the other for decryption. The most important characteristic of this algorithms is:

> **It is computationally infeasible to determine the decryption key given only knowledge of cryptographic algorithm and the encryption key.**

Some algorithms, such as *RSA*, have also another important feature:

> **Either of the two related keys can be used for encryption, with the other used for decryption.**

The essential steps for using the algorithm are:

1. Each user generates a pair of keys.
2. Each user places one of the two keys, **the public key**, in a public register (a place accessible to everyone). The other key is kept private.
3. If a user wishes to send a confidential message to another user, he can encrypt the message with the public key of the receiver.
4. At this point, the receiver decrypts the message using his private key. No other user can decrypt this message because the decryption key is secret.

So, as long as a user's private key remains protected and secret, incoming communication is secure.

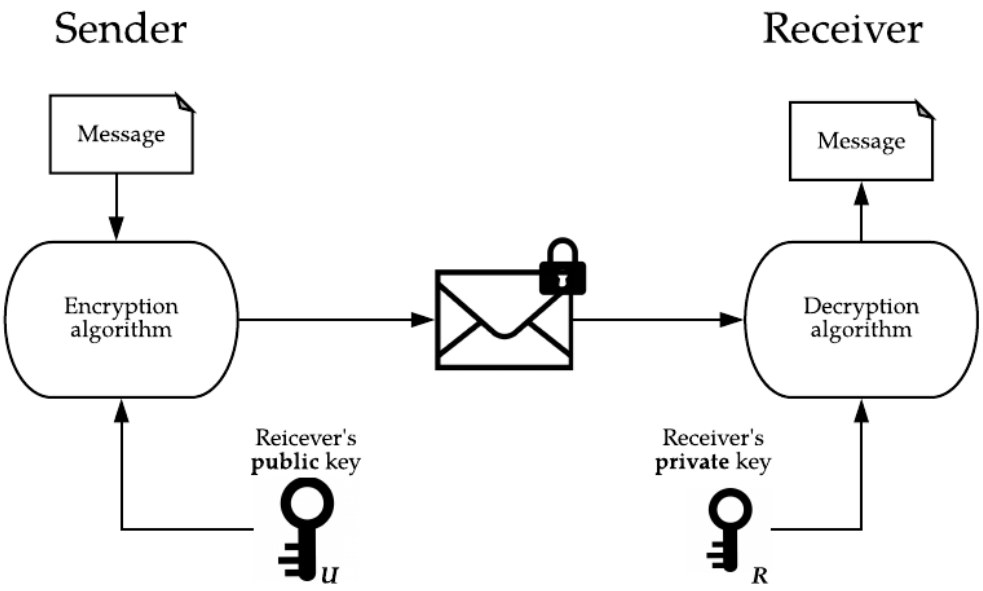

Figure 2.1: Use of public-key cryptosystem for secrecy

This kind of algorithm can not be used in all case due to its heavy computational cost and complexity, so they can not replace symmetric encryption, but depending on the application, the users can use the keys in different way to perform different types of cryptographic function. We can classify the use of public-key cryptosystems into three categories:

- **Encryption/Decryption**: the sender encrypts a message with the recipient's public key. This protects the **confidentiality** of the message from possible interceptions.
- **Digital Signature:** the sender *signs* a message with his private key. The encryption is applied to the whole message or to a small block. In this way the sender is **authenticated** because he is the only one who can create such a signature.
- **Key Exchange**: two parties cooperate to exchange a session key.

Some algorithms are suitable for all uses, such as *RSA*, others are specific to one, such as *Diffie-Hellman* that is designed only for key exchange.

2.2 Basic of digital signature

As we mentioned earlier, there are many algorithms that have the property that both keys can be used for encryption, with the other being used for decryption. This enables another usage schema that provides a different service other than a simple encoding of the message.

The sender encrypts the message with his private key before transmitting it. The receiver decrypts the received message using the sender's public key. In this way, the entire encrypted message serves as **digital signature**. Only the authentic sender could have prepared this message, because his private key is secreted and only he knows it.
In addition, with this protection is impossible to alter the message without access to the private key of the sender, so the message is **authenticated** both in terms of source and data.

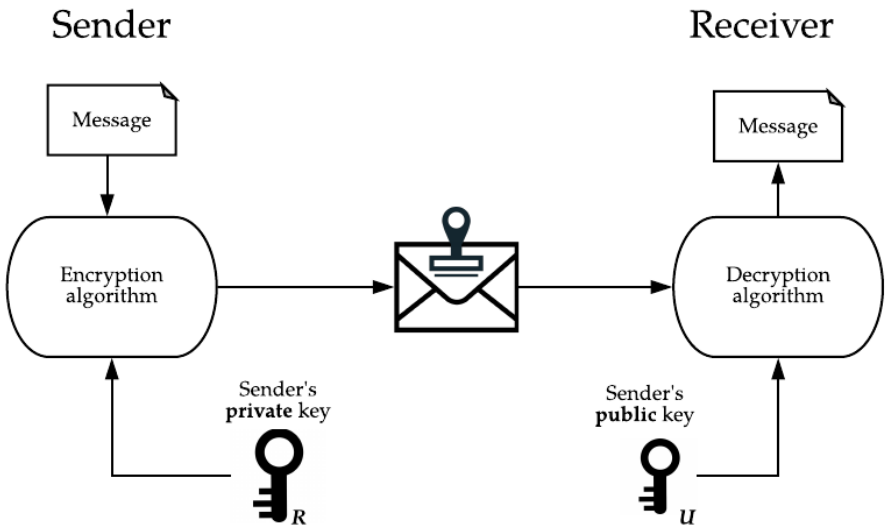

Figure 2.2: Use of public-key cryptosystem for authentication

But this scheme presents two big problems. First, in this way an attacker can easy capture the message and he can decrypt it because the public key of the sender is public and easily recoverable, therefore communication is no longer protected. This problem can be fight with a double use of the public-key schema. The sender encodes the message with his private key, for digitally "sign" the document, and

then he encrypts the signed message with receiver's public key. This schema provide both authentication and confidentiality. This approach requires to use the public key algorithm four times for each communication and this slows down the performance. Second, the encoding of the entire message, for validating author and contents, requires a great deal of storage. A more efficient way of achieving the same results is to encrypt only a "special" block of bits that is a function of the document. Such block is called **authenticator**, and it has the property that is impossible to edit the document without changing the authenticator. If this block is encoded with the sender's private key, it can be used as a signature to verify origin and content.

Usually the authenticator is obtained using an hash function. An **Hash function** is a function that accepts a variable length block of data as input and produces a fixed-size output. This function has the property that the results of applying the function to a large set of inputs will produce output that are evenly distributed and apparently random so, slight differences in input data producing very big differences in output data. This kind of functions, needed for security application, are also called "**cryptographic hash function**". These functions have specific security properties:

- It is considered practically impossible to recreate the input message from its hash value (*Preimage resistant*);
- It is considered practically impossible to compute from a specific message a second message that has the same hash value (*Second preimage resistant or weak collision resistant*);
- It is considered practically impossible to find two different messages that would lead to the same hash value (*Collision resistant or strong collision resistant*).

They are usually used to check if the data has changed because any possible change in the input will imply a change in the resulting hash value. Before sending a message, an hash of the message is precalculated and sent along with it. At destination, the recipient can recalculate the hash of the message and compare it with the one he received from the sender. If the two values correspond, the message has not been altered. Obviously also the hash function must be transmitted in a secure fashion. It must be protected so that if an adversary alters or replace the message, he can not also alter the hash value to fool the receiver.

Integrity services can be combined with authentication services using both hash mechanisms and public-key cryptography. The hash value of a message is encrypted with a user's private key, in order to create a digital signature. In this way, anyone who knows the user's public key can verify the **integrity** and the **authenticity** of the message.

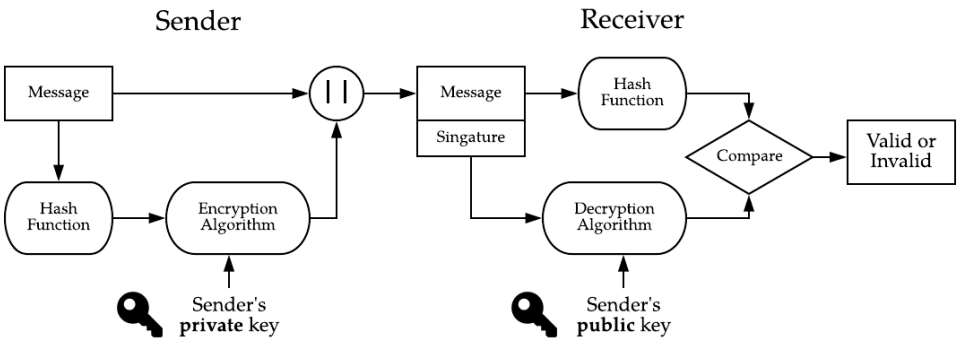

Figure 2.3: Simplified example of Digital Signatures procedure

If also **confidentiality** is desired, then the message plus the signature (private key encrypted hash code) can be encrypted using some cryptographic system, for example using a symmetric secret key. The following image shows the different ways of using public-key cryptography and services that can be obtained (The user ***a*** sends a message to the user ***b***, ***PU*** and ***PR*** are the public and private keys of the users):

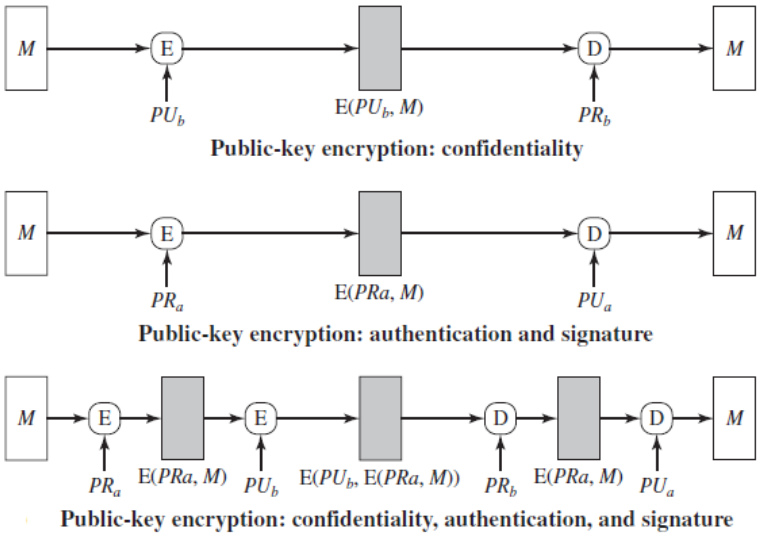

Figure 2.4: Message Encryption with public key - image from [1]

2.3 X.509 Certificate and Certificate Authority

Each user must be able to access the public keys of his interlocutors in order to verify their signatures and to authenticate the messages he received. The access to this data must be protected and guaranteed in order to prevent tampering and fraud. To address this problem were introduced the Certificates and the Certificate Authorities (CA). The binding of a public-key to an entity is provided by an authority through a digitally signed data structure called a **public key certificate** issued by an authority called **Certificate Authorities** or **Certification Authority (CA)**.

The *ITU-T recommendation X.509* [2] define the standard for public key certificates and the information included in it. Some of the most common fields in certificates are:

- **Version**: which X.509 version applies to the certificate (which indicates what data the certificate must include).
- **Serial number**: the identity creating the certificate must assign it a serial number that distinguishes it from other certificates.
- **Signature algorithm information**: the algorithm used by the issuer to sign the certificate.
- **Issuer name**: the name of the entity issuing the certificate (usually a CA).
- **Period of validity**: start/end date and time.
- **Subject name**: the name of the identity to whom this certificate refers.
- **Subject public key information**: the public key associated with the identity.
- **Extensions**: a set of one or more extension fields. They were added in version 3 to allow the insertion of new data within the certificates but without constraining them in fixed fields

A user can present his or her public key to the authority in a secure manner and try to obtain a certificate. If the CA will be satisfied of the identity of a subject and will be sure to not issue public-key certificates for two different subjects with the same subject name, it will create the certificate. The user can then publish

the certificate and anyone needing this user's public key can obtain the certificate and verify that it is valid and signed by a trusted authority. This scheme have the following requirement:

- Any participant can read a certificate and determine the name and the public key of the certificate owner.
- Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
- Only certificate authorities can create and update certificates.

CAs can also revokes a previously issued public-key certificate, so entities need to be able to learn that revocation has occurred in order not to use an untrustworthy public-key certificate. For this reason, the CAs have to maintain *revocation lists* (CRLs).

Also the signature of the CA have to be authenticate, so they also need a certificate. Indeed, there are two primary types of public-key certificates, **end-entity public-key certificates** and **CA certificates**. An end-entity public-key certificate is a public-key certificate issued by a CA to an entity acting as a PKI[1] end entity that cannot use the corresponding private key to sign other public-key certificates. A CA certificate is a public-key certificate issued by a CA to an entity that is also acting as a CA and therefore it is capable of issuing and signing public-key certificates. For this reason, CAs are connected to each other in order to exchange user certificates and authenticate the CAs themselves.

2.4 Properties and legal value of digital signature

With the adoption of the EU Regulation n. 910 of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market, known as "**eIDAS**" (***e****lectronic* ***ID****entification,* ***A****uthentication and trust* ***S****ervices*), the regulatory framework defined by the European Directive 1999/93/EC on electronic signatures and the relevant national transposition laws is now close to a

[1]**Public Key Infrastructure**. RFC 4949 (Internet Security Glossary) defines public-key infrastructure (PKI) as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography. The principal objective for developing a PKI is to enable secure, convenient, and efficient acquisition of public keys..

fundamental update on a European scale, aimed at to guarantee full interoperability at community level of electronic signatures, trust service provider, identification and authentication services. "eIDAS" has created standards for: *electronic signatures*, *qualified digital certificates*, *electronic seals*, *timestamps*, and other proof for authentication mechanisms that enable electronic transactions, with the same legal standing as transactions that are performed on paper. It entered into force on 17 September 2014 and applies from 1 July 2016 except for certain articles. All organizations delivering public digital services in an EU member state must recognize electronic identification from all EU member states.

Focus on the digital signature, eIDAS has developed its own eSignature classifications based on trust and level of assurance provided.

Basic Level Electronic Signatures: according to eIDAS, at the basic level, an electronic signature can be defined as:

> *" 'electronic signature' means data in electronic form which is attached to or logically associated with other data in electronic form and which is used by the signatory to sign "*
> *« [3] eIDAS - Article 3 Definitions (10) »*

The data is in electronic form and attached to other electronic data, but there are problems with this model which eIDAS is trying to address. Firstly, there is no way to tell, with certainty, that the file/document has not been tampered and secondly, there is no way of knowing the true identity of the person who has signed the document. To solve these problems the following classification has been developed.

Advanced Electronic Signatures (AdES):

> *" An advanced electronic signature shall meet the following requirements:*
>
> - *it is uniquely linked to the signatory;*
> - *it is capable of identifying the signatory;*
> - *it is created using electronic signature creation data that the signatory can, with a high level of confidence, use under his sole control;*
> - *it is linked to the data signed therewith in such a way that any subsequent change in the data is detectable. "*
>
> *«[3] eIDAS - Article 26 Requirements for advanced electronic signatures»*

In order to satisfy all of these requirement, you can use **digital signatures based on PKI**. Digital signatures are applied with a *digital certificate* that is only issued after complete verification of your identity by a trusted third party (called Certificate Authority or CA). Digital Certificates and their resulting signatures, are uniquely linked to the individual and virtually impossible to spoof. Because the signatory is the sole holder of the private key, which is used to apply the signature, you can be assured that the signer is the person who he say he is. Finally, part of the signature verification process includes checking if any changes have been made to the document since it was signed.
Advanced electronic signatures can be technically implemented following the XAdES, PAdES, CAdES or ASiC (Associated Signature Containers) standards for digital signatures.

Qualified Electronic Signatures (QES):

> *" 'qualified electronic signature' means an advanced electronic signature that is created by a qualified electronic signature creation device, and which is based on a qualified certificate for electronic signatures "*
> *« [3] eIDAS - Article 3 Definitions (12) »*

A **qualified electronic signature creation device** is a device that meet specific requirements[2]:

1. The **device** must ensure:
    a. The confidentiality of the electronic signature creation data;
    b. The electronic signature creation data used for electronic signature creation can practically only occur once;
    c. The electronic signature creation data used for signature creation cannot be derived and the signature is protected against forgery using current available technology;
    d. The electronic signature creation data used for signature creation can be reliably protected by the legitimate signatory against use by others.
2. The device shall not alter the data to be signed or prevent such data from being presented to the signatory prior to signing;

[2]from *[3] eIDAS - Annex II: Requirements for qualified electronic signature creation devices*

3. Generating or managing signatory data on behalf of the signatory may only be done by a qualified trust service provider;

4. Without prejudice to point (d) of point 1, qualified trust service providers managing electronic signature creation data on behalf of the signatory may duplicate the electronic signature creation data only for back-up purposes provided the following requirements are met:

    a. The security of the duplicated datasets must be at the same level as for the original datasets;

    b. The number of duplicated datasets shall not exceed the minimum needed to ensure continuity of the service

Therefore, for use a qualified electronic signature is necessary to store the creation and signature data on a highly reliable and assured device.
The next part of the definition says that the data on the device must be based on a "qualified certificate for electronic signatures", which can only be purchased by a CA, as opposed to the definition of "advanced electronic signatures", which do not openly state having to use a digital certificate. These definitions seem a bit vague so that these can stay in line with the technological standards that will come in the future.

From a legal point of view, *Article 25* of the regulation [3] states that:

- *An electronic signature shall not be denied legal effect and admissibility as evidence in legal proceedings solely on the grounds that it is in an electronic form or that it does not meet the requirements for qualified electronic signatures.*
- ***A qualified electronic signature shall have the equivalent legal effect of a handwritten signature.***
- *A qualified electronic signature based on a qualified certificate issued in one Member State shall be recognised as a qualified electronic signature in all other Member States.*

Based on what is reported in the regulation, we can state that the digital signature, especially qualified digital signature, have the same legal value of a handwritten signature and guarantees: **data integrity**, **authenticity** and **non-repudiation**.

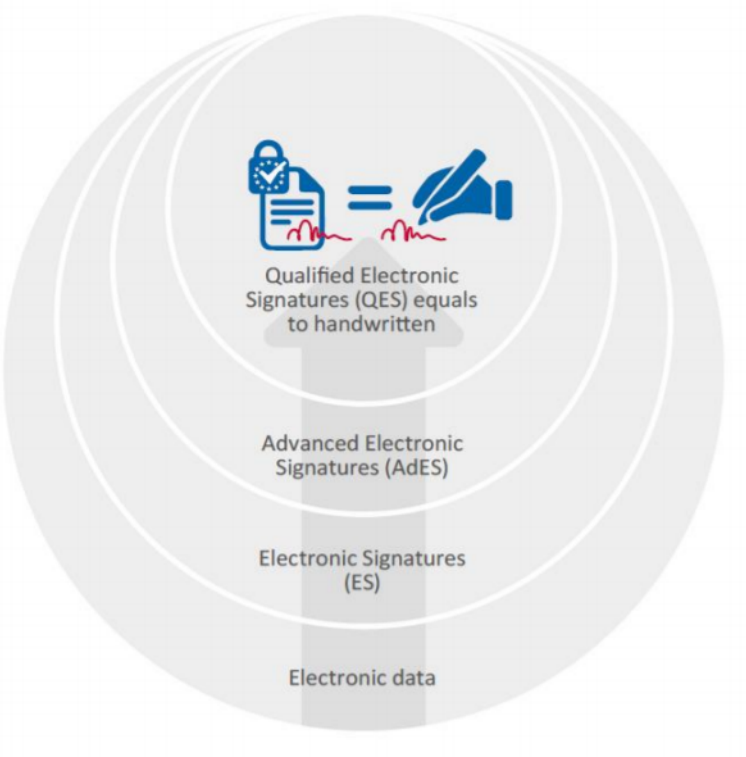

Figure 2.5: eIDAS Signature Type - image from [5]

2.5 Technical standards for digital signatures

As a result of the directive, the ETSI[3] formed an Electronic Signatures and Infrastructures (ESI) technical committee to construct standards for digital signature data structures that meet European requirements for AdES and QES. This has resulted in the development of CAdES, XAdES and PAdES. These standards support different formats suitable for different terms of preservation and security.

2.5.1 CAdES

CAdES (CMS Advanced Electronic Signatures), defined in " *ETSI TS 101 733 V2.2.1. (2013-04) Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAdES)* [10]", is a set of extensions to Cryptographic Message Syntax (CMS) signed data, a basic building block for digital signatures based on public key infrastructure (PKI) principles, making it suitable for advanced electronic signatures defined in the eIDAS regulation. CAdES is a digital signature that can be applied to any type of file. This method generates a "cryptographic envelope",

[3] **ETSI**: European Telecommunications Standards Institute

with a digital signature which guarantees its authenticity and integrity, containing the original document and is characterized by a file with *"p7m"* extension.

In conformance with the specifications, the signer must create their signature in one of two formats: **CAdES Basic Electronic Signature** (CAdES-BES) or **CAdES Explicit Policy Electronic Signature** (CAdES-EPES).

A signature created under **CAdES Basic Electronic Signature** will contain: the signer data (e.g. a document), a collection of mandatory signed attributes and the digital signature value that has been generated on the user data and signature attribute. It may contain also other optional attribute.

CAdES Explicit Policy-based Electronic Signature extends the definition of an electronic signature to conform to the identified signature policy. A *signature policy* is a set of rules for the creation and validation of electronic signatures that defines the technical and procedural requirements for their creation, validation and long-term management in order to satisfy particular business needs and to determine their validity. A CAdES-EPES incorporates a signed attribute (sigPolicyID attribute) indicating the signature policy that shall be used to validate the electronic signature.
The -BES and -EPES formats have been incorporated into the new -B format.

The validation of an electronic signature requires additional data that can be added to both CAdES-BES and CAdES-EPES format. As reported in the technical specification[10], this additional data is called **validation data**, and includes:

- Public Key Certificates (PKCs).
- Revocation status information for each PKC.
- Trusted time-stamp in addition to default time-mark.
- Information regarding signature policy used to verify electronic signature.

Dependent upon the above validation data chosen, four standard CAdES formats were specified:

- **CAdES-B**: Basic Electronic Signature, the simplest version. This level combines the old -BES and -EPES levels;
- **CAdES-T** : a trusted time is associated with the electronic signature;

- **CAdES-C** : this format adds, to CAdES-T, both *complete-certificate-references* (references to all the certificates present in the certification path used for verifying the signature) and *complete-revocation-references* (references to the CRLs[4] and/or OCSP[5] responses used for verifying the signature) attributes[10].
  CAdES-C can be extended by adding attributes for support long term verification and for preventing some disaster situations. Some format created extending CAdES-C are: CAdES-X Long, CAdES-X Type 1, CAdES-X Type 2, CAdES-X Long Type 1 or 2.).
  Adding one or more long-term-validation attributes to one of this format, we can obtain a **CAdES-LT** (Long-Term form);

- **CAdES-A**: formats under this category allow for backward compatibility and preservation of long-term signature validation for archived documents. Adding one or more long-term-validation attributes to a CAdES-A we obtain a **CAdES-LTA** that is used for archival of long-term signatures.

The old levels: *-BES, -EPES* are no longer used and replaced with *-B* and the *-C* is replaced with *-LT* levels.
For a complete technical description of CAdES standard: [10]*ETSI TS 101 733 V2.2.1 (2013-04).*

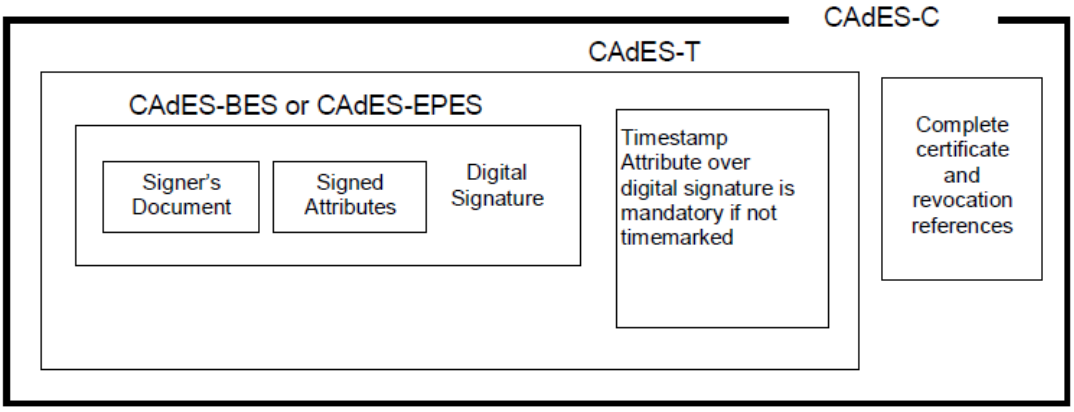

Figure 2.6: Illustration of CAdES-C format

[4]**Certificate Revocation List**: a list of digital certificates that have been revoked by the issuing certificate authority (CA) before their scheduled expiration date and should no longer be trusted.

[5]**Online Certificate Status Protocol**: an Internet protocol used for obtaining the revocation status of an X.509 digital certificate.

2.5.2 XAdES

XAdES (XML Advanced Electronic Signatures), is a set of extensions to XML-DSig[6] recommendation making it suitable for advanced electronic signatures. W3C and ETSI maintain and update XAdES together. While XML-DSig is a general framework for digitally signing documents, XAdES specifies precise profiles of XML-DSig making it compliant with the European eIDAS regulation. The digital signature in this format is mainly used on XML files and only rarely for other files. Like CAdES, also the XAdES documentation defines Basic Electronic Signatures and Explicit Policy Electronic Signatures and describes many profiles (forms) differing in protection level offered.

For more details: [11]*ETSI TS 101 903 V1.4.2 (2010-12).*

2.5.3 PAdES

PAdES (PDF Advanced Electronic Signatures) articulates the same capabilities featured in CAdES and XAdES for PDF. Its technical specification is published by ETSI as *"TS 102 778"* and it is divided into 6 parts. PAdES differs from CAdES and XAdES in that it applies only to PDF documents and defines requirements that PDF viewing and editing software must follow when using digital signatures in PDF documents. The standard also defines how a signature can be displayed, at a particular position on a particular page, and how digital signatures can be integrated with the form-filling features of PDF. This is a key factor that distinguishes it from the other two standard, which are more suited for applications that may not involve human-readable documents. Also in this case the standard define different profiles that offers different protection level.
With PDF, the signature data is incorporated directly within the signed document and this allows the complete self-contained PDF file to be copied, stored, and distributed as a simple electronic file.
Signing PDF documents has broad application for the following reasons:

- PDF is used in much the same way as paper documents and it provides a visual representation of a document;
- PDF providing a visual appearance of the signature. Multiple signatures within documents are also supported, complete with visual appearances.

[6]XML Signature (also called XMLDSig, XML-DSig, XML-Sig) defines an XML syntax for digital signatures

- It is possible to add information directly in the document with the PDF fillable form features.
- PDF can be used as packaging mechanism for holding other documents by making them attachments to a parent PDF document. A signature on the parent PDF file will make it possible to detect changes to any attachments contained in the PDF file.

Today, end-to-end electronic workflows involving signed documents are implemented based on PDF and its digital signatures [8].

Chapter 3

Java Command Line Tool for Digital Signature

The aim of this thesis is to facilitate and automate the use of digital signature. In particular, the qualified electronic signature was taken into consideration. The signature is performed with a cryptographic token that complies with the PKCS#11 standard.
The PKCS#11 standard defines a platform independent API, also called "Cryptoki", for devices that hold cryptographic information and perform cryptographic functions, such as hardware security modules (HSM) and smart cards.
Cryptoki follows a simple object based approach, addressing the goals of technology independence (any kind of device) and resource sharing (multiple applications accessing multiple devices), presenting to applications a common, logical view of the device called a "cryptographic token" [13]. The API defines most commonly used cryptographic object types (RSA keys, X.509 Certificates, etc.) and all the functions needed to use, create/generate, modify and delete those objects.
A USB smart card reader[1] and an Italian CNS ("Carta Nazionale dei Servizi"), provided by Aruba, were used to test and develop the tool.

Figure 3.1: Bit4id smart card reader

[1]Bit4id miniLector S EVO, `https://www.bit4id.com/en/lettore-di-smart-card-minilector-s-evo/`

The "Carta Nazionale dei Servizi" (CNS) is the tool that allows the identification of the user (Certificate Holder) for access to online services of the Public Administration, such as the services available on the website of the "Agenzia Delle Entrate", "INPS", "Equitalia" etc. The CNS Certificate is issued on a Smart / SIM Card support, together with the digital signature certificate.

Figure 3.2: Carta Nazionale dei Servizi

3.1 Command Line Tool

The developed software is a **command line tool** that interacts with the token and it allows to sign documents respecting the CAdES or PAdES standard. The choice of this type of software is due to their versatility and speed of use and the possibility to easily integrate them into other software.

The tool have two fundamental commands that are used for select the type of signature to execute: *cades* or *pades*. Both commands take as primary argument the file to sign. The tool supports also some generic options for get information about certificates and keys stored in the token.

-i, - -info-certificates: it shows information about certificates stored in the token/smartcard.

-u, - -key-usage: it shows data of *KeyUsage* extensions. Key usage extensions define the purpose of the public key contained in a certificate. A user can use them to restrict the public key to as few or as many operations as needed. For example,

if the user has a key used only for signing or verifying a signature, he can enable only the digital signature and/or non-repudiation extensions.

The system is already equipped with the necessary driver for interfacing with the USB token used. In case the user wants to use another driver, to support a different signature device, he can specify it using the " **-d, - -driver** " option. If the supplied driver file does not exist, the system attempts to use the default driver.

3.1.1 CAdES Command

The command allows the user to create a signature in CAdES format. The options of the command are:

```
cades: CAdES signature format
  Usage: cades [options] FileToSign
    Options:
      -h, --help
        display help
      -c, --choose-certificate
        choose certificate to use
      -o, --output-folder
        set destination FOLDER for the output file
      -n, --newfile-name
        set name of the new file
```

The tool accesses to the token, using the provided password, it signs the document and it creates a new signed file with the same name of the original but with *p7m* extension. The tool automatically selects a certificate with "non-repudiation" keyUsage, but if the token contains multiple certificates that can be used to sign, you can use the *"-c"* option to choose which one to use. With the options *"-o"* and *"-n"* the user can specify the output folder and the name of the new file.

3.1.2 PAdES Command

The command allows the user to create a signature in PAdES format. The possible parameters of the command are:

```
pades:   PAdES signature format
   Usage: pades [options] FileToSign
     Options:
       -h, --help
         display help
       -c, --choose-certificate
         choose certificate to use
       -o, --output-folder
         set destination FOLDER for the output file
       -n, --newfile-name
         set name of the new file
       -v, --visible-signature
         add visible signature - only text
         Default: false
       -vi, --visible-signature-image
         add visible signature - text and image
       -f, --field-to-sign
         name of the field to sign
       -pg, --page
         page of signature
         [If a field is selected this option is ignored]
         Default: 1
       -pv, --vertical-signature-position
         vertical position of visible signature:
        T(op) - M(iddle) - B(ottom)
         [If a field is selected this option is ignored]
       -ph, --horizontal-signature-position
         horizontal position of visible signature:
        L(eft) - C(enter) - R(ight)
         [If a field is selected this option is ignored]
```

In this case the created file is a signed PDF. For PAdES signature we can distinguish between visible and not visible signature.

For **not visible signature** the behaviour of the command is similar to the "cades" command and it is enough to pass the file to sign and the password to access to the token.

For **visible signature** several parameters are needed to place the signature in the desired place. First is needed to activate visible signature using *-v* or *-vi* option. The first inserts only a text signature, while the second allows the user to add an image to be associated with the signature. At this point, if the PDF contains some signable fields, the tool shows a list of all usable fields and the user can select one of these or skip this step. If the user does not use a field, he can place the signature at a specific position using some anchors points.
The options: *-pg*, *-pv* and *-ph*, allow the user to choose the page and the horizontal and vertical position.
The possible anchor values for **vertical position** are: *T(op)*, *M(iddle)* and *B(ottom)*.
For **horizontal position** the anchor values are: *L(eft)*, *C(enter)*, *R(ight)*.
It is possible to provide other information to the tool to override the default behaviour, such as: the name of the new file to be created, the output folder and the name of the field to be used for signing. Like the cades command, also with this, it is possible to choose the certificate to use for signing with the option *-c*.

3.2 Software design and implementation

The software was developed in Java 8 (with Oracle *JDK8*) and it is tested on *Windows 10* and *Windows 8.1* with the signature creation device mentioned above. The tool allows to create signatures in *CAdES* and *PAdES* format and for both it uses the *-B* signature level and *SHA-256* as digest algorithm.

The software operations are very simple. It receives in input, via command line arguments, the options for the signature, as the type of signature and the options associated with it and the file to sign. Once checked that the received parameters are acceptable, the necessary structures are created, the software connects to the token and it sends to the device all data necessary for the creation of the signature. Once the procedure is finished the software creates a new signed document.

The following flow chart shows the execution of the actions performed by the software.

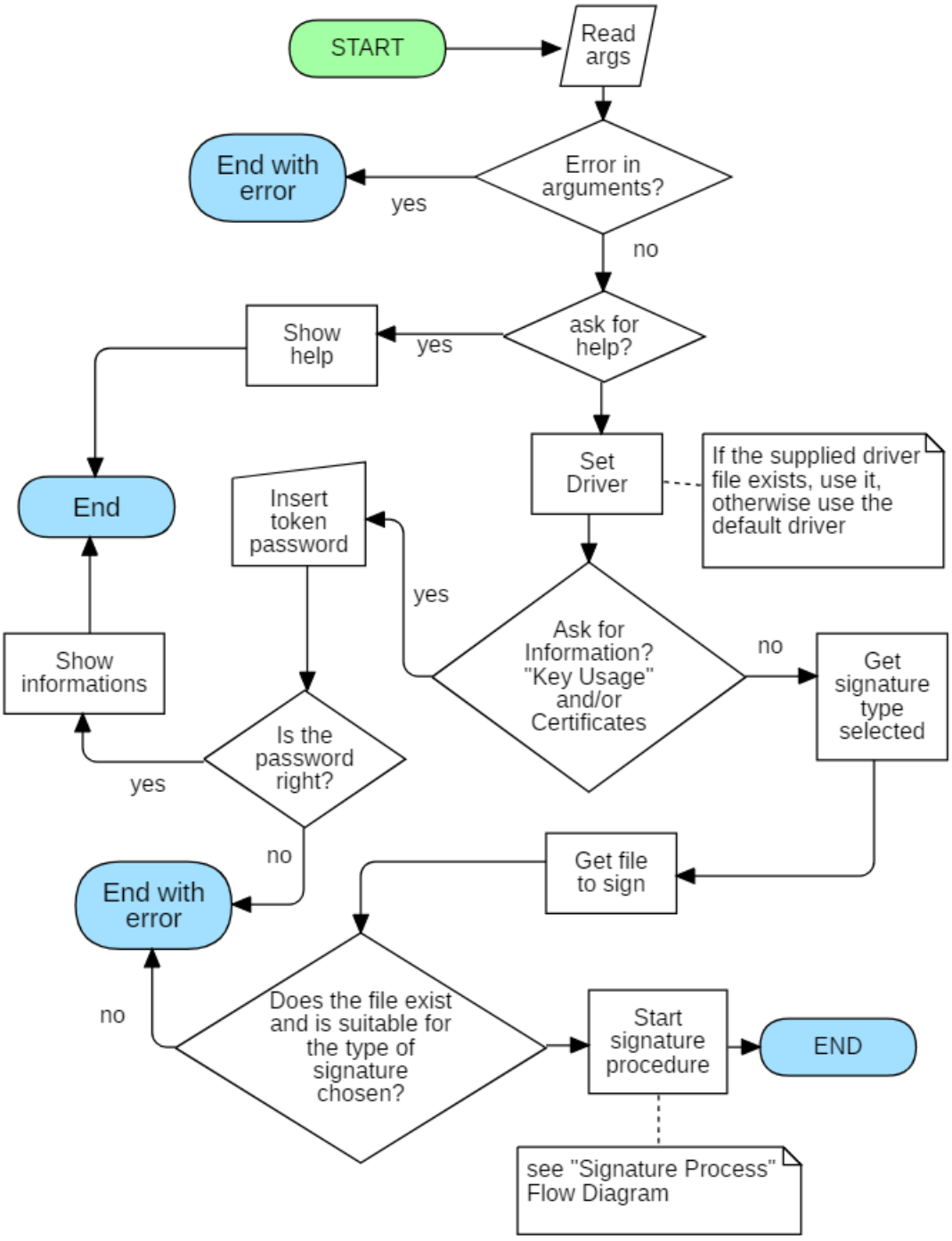

Figure 3.3: *Flow chart* of main process

The software uses two external libraries to achieve its goal: **JCommander** and **Digital Signature Services (DSS)**.

JCommander: this library is a very small Java framework that makes the analysis of command line parameters trivial. It allow to descibe the desired options using some Java annotations. In this way it is very easy and fast to define many argument for a command line tool. The library supports both simple arguments and complex commands with many secondary arguments, such as the cades and pades command used by the tool. Some examples:

```
public class Args {
      @Parameter(names = { "-h", "--help" }, description =
          "display help", help = true)
      public boolean help = false;
      @Parameter(names = { "-t","--text" }, description =
         "pass text", required = false, arity = 1)
      public String text;
}
```

At this point it is sufficient to use JCommander to analyze the list of arguments and it will take care of filling a pojo[2] containing the options structure.

```
class JCommanderTest {
     @Test
     void test() {
          Args args = new Args();
          String[] argv = {"-h","-t","hi"};
          JCommander.newBuilder()
            .addObject(args).build().parse(argv);
          Assert.assertEquals(args.help, true);
          Assert.assertEquals(args.text,"hi");
     }
}
```

[2]A **Plain Old Java Object (POJO)** is an ordinary Java object, not bound by any special restriction. In this case, the only exception is represented by the JCommander specific annotations, necessary to associate the arguments with the POJO properties.

Digital Signature Services (DSS): it is an open-source software library, developed by the *CEF* (Connecting Europe Facility)[3], for electronic signature creation and validation. DSS supports the creation and verification of interoperable and secure electronic signatures in line with European legislation. In particular, DSS aims to follow the *eIDAS Regulation* and related standards [15]. DSS allows re-use in a variety of different ways: in an applet, in a stand-alone application or in a server application. The library is open-source and released under LGPL 2.1 license.

The DSS framework performs three atomic steps to sign a document:

1. Compute the digest to be signed;
2. Sign the digest;
3. Sign the document (add the signed digest to the document).

First, we need to specify the type of *KeyStore* to use for signing the document, in other word, where to private key can be found. The framework offers different implementations, but given the use of a PKCS#11 token, we use *Pkcs11SignatureToken* implementation that allows to communicate with a token using PKCS#11 interface. It requires a driver, which provides a specific implementation of the PKCS#11 interface for the device used, to create the connection. Synthetically, to sign a document we need to follow some steps[4]:

1. Create an object based on *SignatureParameters* class. Generally, the number of parameters depends on the profile and type of signature. For example, the parameters for a PAdES signature may contain also information about the image and the text to show in the visible signature.

2. Choose the profile, the packaging[5] and signature digest algorithm. The encryption algorithm is determined by the private key and therefore it can not be forcibly set in the object that contains the signature parameters.

[3]**The Connecting Europe Facility** (CEF) is a European Union fund for pan-European infrastructure investment in transport, energy and digital projects which aim at a greater connectivity between European Union member states

[4]Process description taken from the official framework documentation, *DSS Cookbook* [16].

[5]When signing data, the resulting signature needs to be linked with the data to which it applies. This can be done either by creating a data set which combines the signature and the data or placing the signature in a separate resource and having some external means for associating the signature with the data. So, we need to define the packaging of the signature, namely ENVELOPED, ENVELOPING or DETACHED [16].

3. Indicate the private key entry to be used. Once the connection with the token has been created it is possible to obtain the data of the certificate and so the key and the signer data.

4. Instantiate the adequate signature service. The service used will depend on the type of signature to perform (e.g: *CAdESService* or *PAdESService*).

5. Carry out the signature process. The process of signing takes place in three steps:

   - Retrieve the data which is going to be digested and encrypted. This is performed by the call of *"getDataToSign()"* method on the service object previously instantiated. It gets as input parameter the document to be signed and the *signatureParameter* created at the step (**1**).

```
DSSDocument toSignDocument = new FileDocument(
   inputFile);
ToBeSigned dataToSign = service.getDataToSign(
   toSignDocument, parameters);
```

   - The next step is a call to the function *"sign()"* which is invoked on the object token representing the *KeyStore* and not on the *service*. This method takes three parameters. The first is the array of bytes that must be signed. It is obtained by the previous method invocation (*getDataToSign()*). The second is the algorithm used to create the digest (e.g: SHA256). And the last one is the private key entry obtained in step (**3**) by connecting to the token.

```
DigestAlgorithm digestAlgorithm = parameters.
   getDigestAlgorithm(); //eg. SHA256
SignatureValue signatureValue = token.sign(
   dataToSign, digestAlgorithm, signer);
```

   - The last step of this process is the integration of the signature value in the signature and linking of that one to the signed document based on the selected packaging method. To do this it is enough to call *"signDocument()"* on the service. We have to pass to it three parameters: again the document to sign, the signature parameters and the value of the signature obtained in the previous step.

```
DSSDocument signedDocument = service.
   signDocument(toSignDocument, parameters,
   signatureValue);
```

At this point it is possible to write the signed document to a file.

The signature process follows more or less the same steps for all available formats. The points in which the implementation differs are: the creation of parameters and service. The extension of the tool, to other signature formats such as XAdES, is quite simple because all implementations and creation of objects dependent on the type of signature are encapsulated in a "factory" object (*CAdESSignatureFactory PAdESSignatureFactory*) that extends a common interface (*ISignatureFactory*). With this structure, which follows the *"AbstactFactory"* pattern scheme, it is sufficient to add another factory that implements the necessary methods, in accordance with the signature format to be implemented, and use it in the signature procedure.

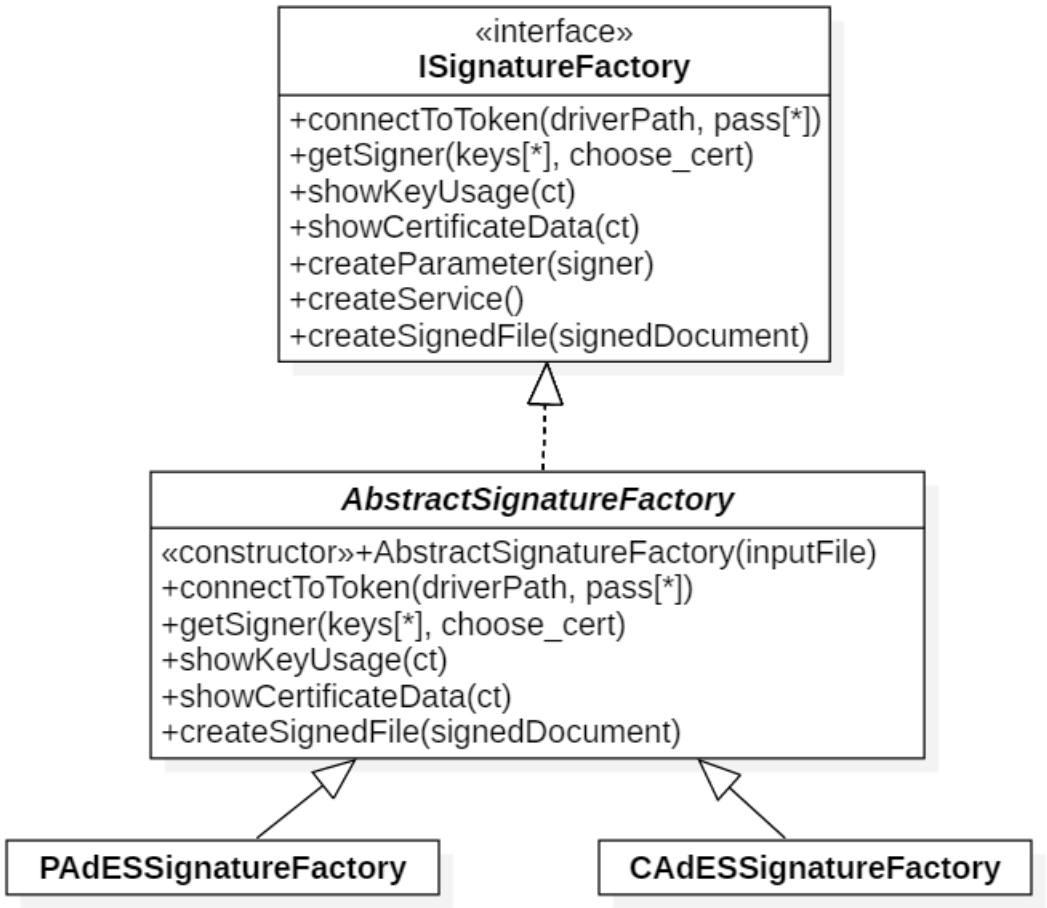

Figure 3.4: Signature Factory Structure

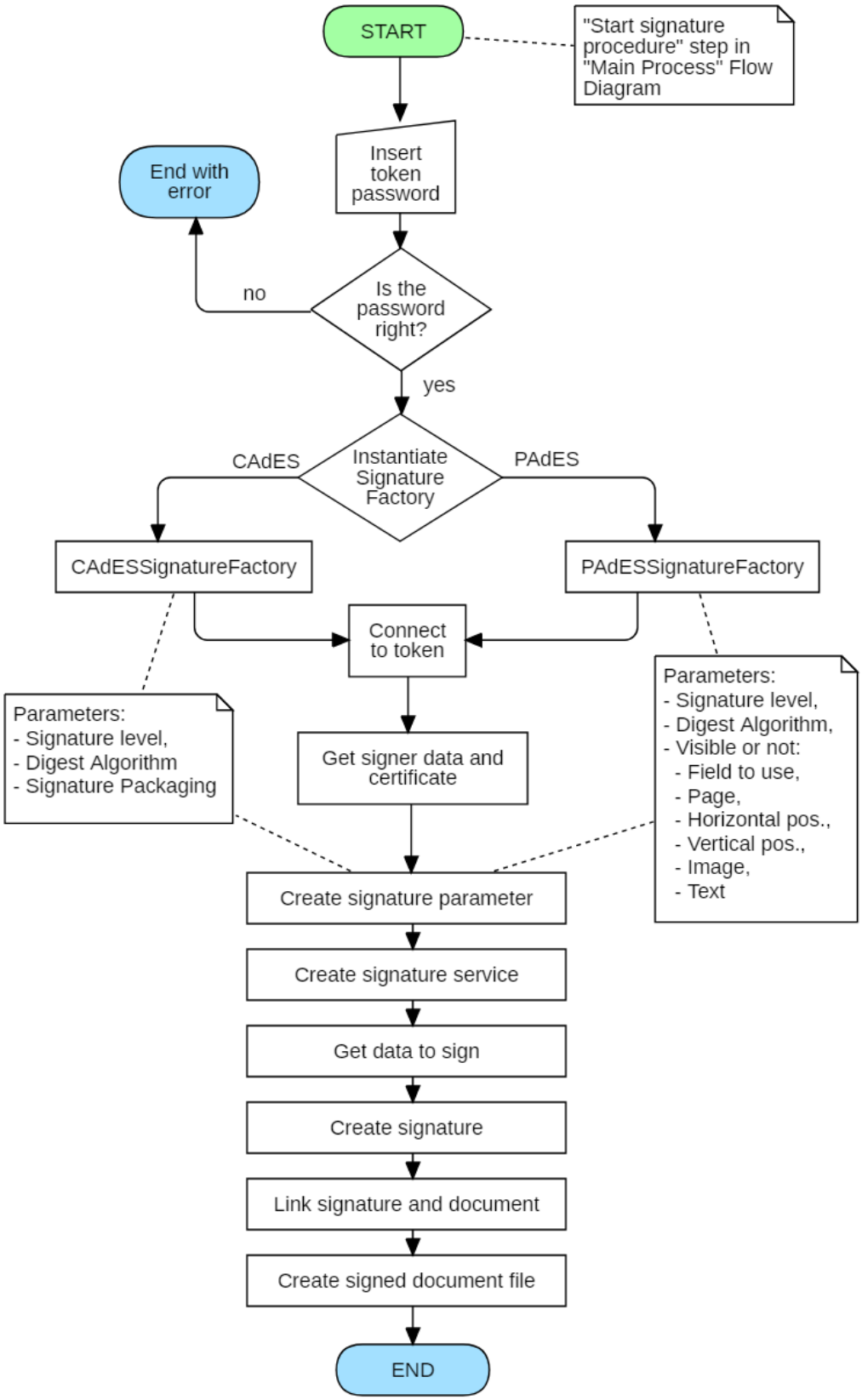

Figure 3.5: *Flow Chart* of signature process

Chapter 4

Integration Digital Signature into a browser

4.1 Browser extension

The use from the command line is convenient and fast but it is not the most congenial and user friendly method. The best way to facilitate and speed up the use of the digital signature is to integrate it, in the most transparent way, into a software used continuously and known by most users, such as a browser. The browser, unlike any desktop application that requires different installation and configuration steps, is already available on virtually all computers and is used consistently by users of all kinds. According to global statistics at the end of 2018, the most used browser is "Google Chrome", which reaches almost 70% of use[1].

Browser	Percentage of use
Chrome	69.56%
Firefox	10.17%
IE	6.02%
Safari	5.63%
Edge	4.22%
Opera	2.30%

Table 4.1: Desktop Browser Market Share Worldwide - **Oct. 2018**

For this reason the decision was to opt for the development of an extension for Google Chrome.

[1]Data from: `http://gs.statcounter.com/browser-market-share/desktop/worldwide`

4.2 Chrome extension structure

The *Chrome Documentation* defines an extension in the following way:

> "**Chrome Extensions** are small software programs that customize the browsing experience adding functionality and behaviour to individual needs or preferences. They are built on web technologies such as HTML, JavaScript and CSS. An extension must fulfill a single purpose that is narrowly defined and easy to understand. A single extension can include multiple components and a range of functionality, as long as everything contributes towards a common purpose."[17]

So, an extension is made up of different, but cohesive, components. These components can include: *background scripts*, *content scripts*, *options page*, *UI elements* and various *logic files*. The components used will depend on the features that you want to implement so they may not be all present. The characteristics of the extension are defined in a file called *manifest.json*.

manifest.json

```
{
 "name": "Digital Signature Tool",
 "version": "1.0",
 "description": "Extension for digital signature with PKCS#11
    token",
 "permissions": ["declarativeContent", "nativeMessaging", "
    activeTab", "downloads", "tabs", "storage"],
 "background": {
   "scripts": ["background.js"],
   "persistent": false
   },
 "page_action": {
   "default_popup": "popup.html"
  },
 "icons": {
   "128": "icon/icon128.png",
   "48": "icon/icon48.png"
  },
 "manifest_version": 2
}
```

Let us analyze the main components of this JSON:

- **"background"**: it registers the background scripts. There is only one background script and it is *not persistent* to prevent a constant consumption of system resources.
- **"page action"**: the extension support only a *"Page Action"*, so its action depends on the current active page in the browser. The activation of the extension is limited only to pages showing a PDF document. This restriction is implemented in the background script using the *"Declarative Content API"*. Here is also defined the *popup* that is shown when the "Page Action" is activated.
- **"permissions"**: it lists the permissions needed to access the browser functionality and the Chrome API.
  - **declarativeContent**: the *"Declarative Content API"* allows to show or block page action depending on the URL of web pages.
  - **nativeMessaging**: it allows to use the *"Native Messaging API"* to exchange messages with native applications on the system.
  - **activeTab** and **tabs**: they allow to interact with the browser's tab system and temporary access to the currently active tab.
  - **downloads**: it allows the extension to use *"Downloads API"* for programmatically initiate, monitor, manipulate, and search for downloads.
  - **storage**: it enables the use of *"Chrome storage"* to store and retrieve data.

The main problem to solve, for the development of the extension, is to create a communication channel between the browser and the signature device. The browser, for security reasons, acts in a sort of sandbox and can only access some specific hardware like camera or microphone. The problem can be solved by putting in communication, not directly the extension with the token, but allowing an exchange of messages between the extension and a **native application**, that it is normally runnable on the system. The native application will have the task of interacting with the token and performing the operations of signature. This is made possible by the *"Native Messaging API"* offered by Google. The native application must register itself as a *"native application host"* in order to allow the extension to find it

and start a communication. This is done by creating a *manifest file* (different from previous manifest file, this is defined for the native application), which defines the configurations of the application and it specifies the extensions enabled to communicate with it. It is also needed to add some keys to the system registry in order to allow Chrome to find and start the host app in a separate process and communicate with it using standard input and standard output streams[2]. In our case, the native application is the previously developed **Java command-line tool**. The tool requires some minor changes to send and read messages from the extension and to adapt the received data in the supported format.

The main elements that compose the extension are:

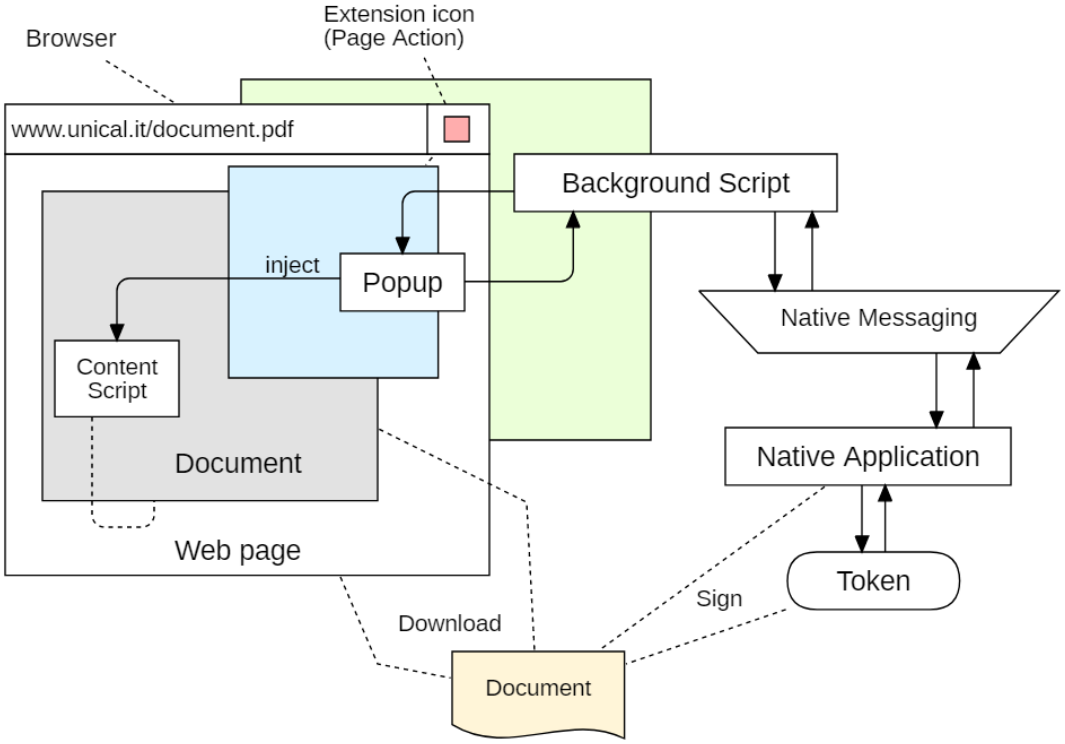

Figure 4.1: Interactions and components of the extension

- **Popup page**: it is composed of an HTML page, which constitutes the user interface of the extension and a JavaScript script that implements its logic. It allows the user to set: the type of signature, some parameters necessary for the visible PAdES signature and the password to access the token. Once ready, it communicates the data to the *"Background Script"*.

[2]For more detailed information and example: `https://developer.chrome.com/extensions/nativeMessaging`

- **Background script**: it stores the data entered by the user and it is responsible for communicating with the native application.
- **Content Script**: it is used only in case of visible PAdES signature. The script is injected into the page and modifies the Chrome PDF viewer to show the names of the fields that can be used for a signature. It tries to position the labels with the field name as accurately as possible, adapting their positions also based on the browser zoom level. To do this, it must manipulate the dimensions of the *"embed"* object, which contains the PDF viewer, and the *"body"* of the page, to manage the size and orientation of the various pages that may be contained in the file.
- **Native application**: a Java application that deals with the creation of the signature and communication with the token. The native application communicates exclusively with the *"Background Script"* and it responds to two types of requests: **"sign a file"** or **"analyze a file"** to extract information that can later be used to execute a signature (visible PAdES).

4.3 Usage of the extension

The extension can be activated only on a browser tab that contains a PDF. The usage is very simple. The user clicks the extension icon and opens the popup. The first popup screen allows he to choose the type of signature to use, CAdES or PAdES (in this case the user can select if the signature should be visible or not).

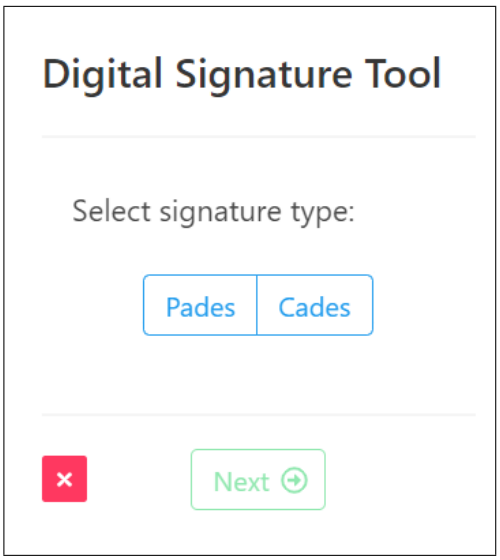

Figure 4.2: Select signature type

CAdES and not visible PAdES: in these cases there are no other special options to set and the popup asks to enter the password and after confirming, the extension will download the PDF (only if necessary) and then it will send all the data to the native application that will sign the document. When the process will be completed, the native application will notify it to the extension and the popup will show a completion message.

PAdES with visible signature: in this case the behaviour is a bit more complex. After the signature type selection the extension will download the file (only if necessary), and it will ask to the native application to analyze the PDF to look for some field suitable for the signature and retrieve some information about the structure of the document. The tool extracts the size and orientation of each page and for each signature field it calculates its position (measured in *points*) on the page. At the end of the analysis, the native application sends the information to the extension and the popup shows a new screen to set the options for the visible signature. The user can choose whether to place the signature in a predisposed field in the PDF (if present) or to use the anchor points and specify a page. If the document contains some signable fields, the extension will inject, in the web page that contain the document, the *"Content Script"* that will modify the appearance of the PDF viewer to add the names of the fields "above" the viewer. After filling in the form with the signature options, it is possible to enter the password and execute the signature.

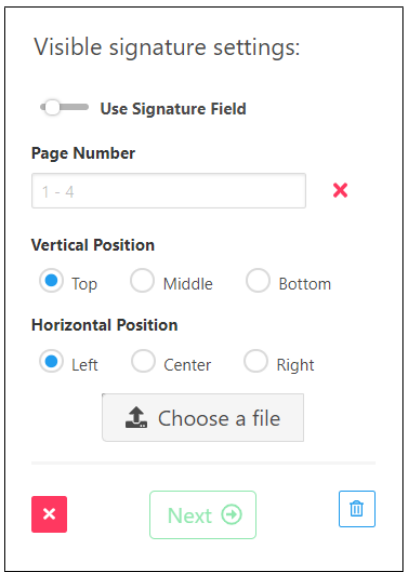

Figure 4.3: Use anchor points

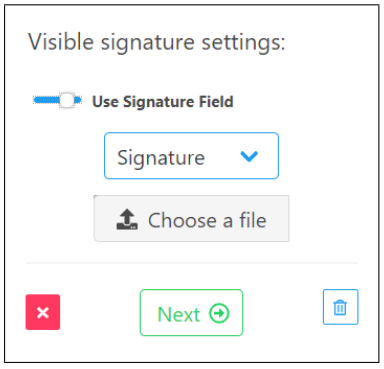

Figure 4.4: Use signature field

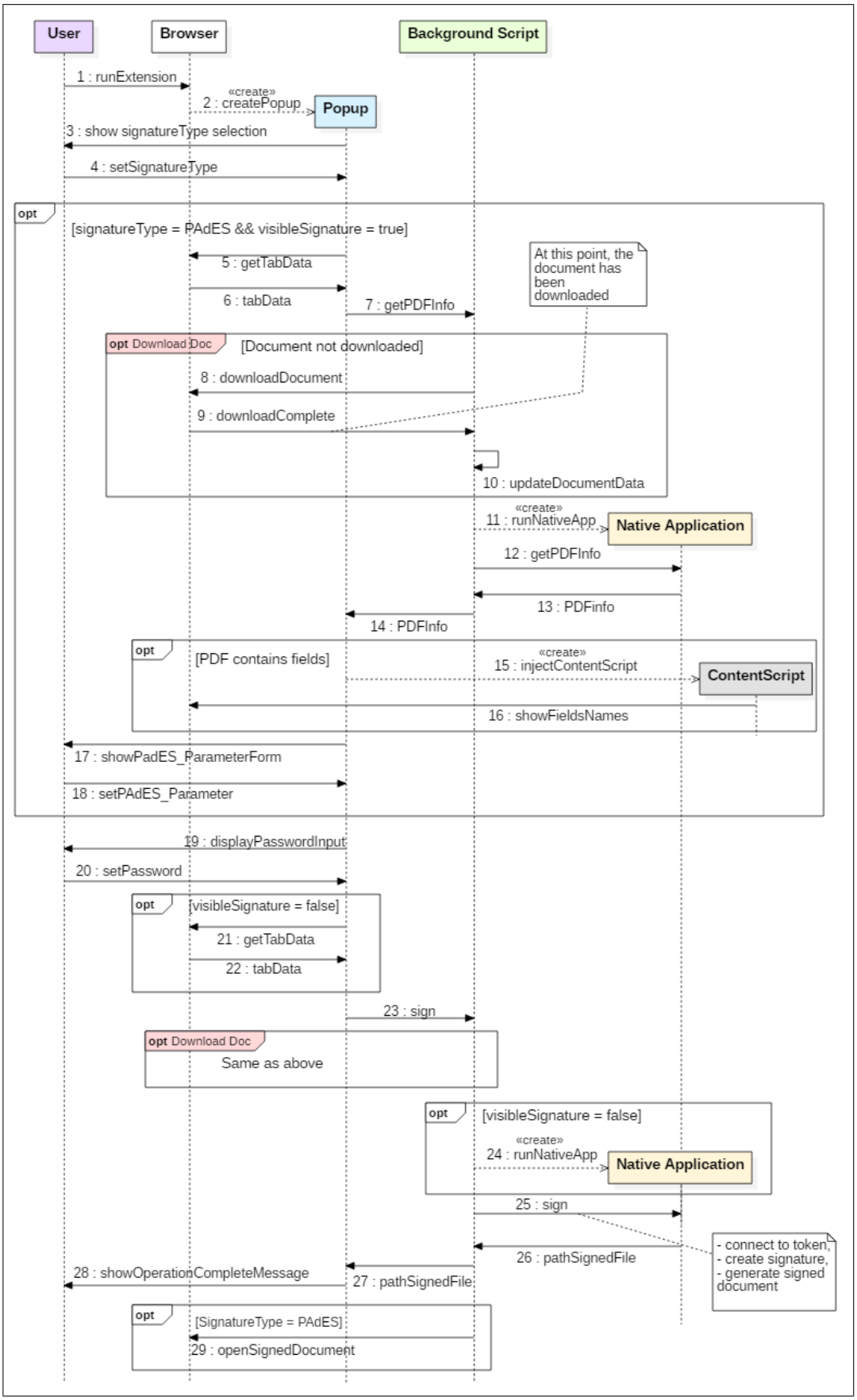

Figure 4.5: *Sequence Diagram*: interaction between extension components

Chapter 5

Conclusions

In an attempt to speed up and automate the operations related to the management and creation of digitally signed documents, this thesis work has led to the development of two software modules that in a different way can fulfill, at least in part, this purpose.

The first, with its use via command line, allows faster use than an application with a graphical interface, that requires several steps and interactions with the user, and it is easy to integrate into other programs that can invoke it directly, such as a script *Python* or *Perl*. The tool supports only two signature formats, *CAdES* and *PAdES*, but these are the most used. The addition of the support to other formats, such as the *XAdES* format, is easily achievable as previously mentioned.

The second, the extension for the Chrome browser, is easily to use and user friendly even for those who have no technical knowledge but uses a computer every day as those who could work in an administrative office. The convenience lies in the fact of being integrated directly into the browser and it provides a direct signature method when viewing a document, avoiding the steps of downloading and using other software just to sign. The addition of functionality to it is related to the extension of the first software that is used transparently to the user, but it is a fundamental part because it is the one that is really responsible for creating the signatures.

It would be interesting to extend the project to provide a suite of functions for the general management of signed documents, ranging from their creation to the validation. It is possible to add support to new standard formats or new properties to the signature for increase security, such as *"Trusted Timestamp"* or *"Long-Term Validation"*. Another very useful thing would be the direct connection with other systems necessary for the processing of signed documents, such as a protocol or archiving system, to create an automatic interaction with other system. In this way, for example, a user can open a document on the browser, sign it and send it directly to another system needed for processing, all using only the Chrome extension.

Bibliography

[1] William Stallings.
Cryptography and Network Security: Principles and Practice.
Pearson, ISBN: 978-0133354690, 2003.

[2] ITU-T.
X.509 : Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks.
10/2016.
`https://www.itu.int/rec/T-REC-X.509`

[3] The European Parliament.
Regulation (EU) No 910/2014 of the European Parliament and of the council on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC - (eIDAS).
23/7/2014
`https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2014.257.01.0073.01.ENG`

[4] GlobalSign Blog.
What's the Difference Between Advanced and Qualified Signatures in eIDAS?
`https://www.globalsign.com/en/blog/difference-between-eidas-advanced-and-qualified-electronic-signatures/`

[5] ENISA - European Union Agency for Network and Information Security.
Security guidelines on the appropriate use of qualified electronic signatures.
29/6/2017
`https://www.enisa.europa.eu/publications/security-guidelines-on-the-appropriate-use-of-qualified-electronic-signatures`

[6] AGID - Agenzia per l'Italia Digitale.
Il Regolamento UE n. 910/2014 - eIDAS
`https://www.agid.gov.it/it/piattaforme/eidas/firma-digitale-verso-eidas`

[7] Decreto Legislativo 7 marzo 2005, n. 82
Codice dell'amministrazione digitale (CAD)

- Art. 24. Firma digitale;
- Art. 35. Dispositivi sicuri e procedure per la generazione della firma qualificata

`https://docs.italia.it/italia/piano-triennale-ict/codice-amministrazione-digitale-docs/it/v2017-12-13/`

[8] Adobe Systems Incorporated.
The AdES family of standards: CAdES, XAdES, and PAdES.
`https://blogs.adobe.com/security/91014620_eusig_wp_ue.pdf`

[9] Connecting Europe Facility, e-Signature standards
`https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/e-Signature+standards`

[10] European Telecommunications Standards Institute.
ETSI TS 101 733 V2.2.1. (2013-04) Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAdES)
`https://www.etsi.org/deliver/etsi_ts/101700_101799/101733/02.02.01_60/ts_101733v020201p.pdf`

[11] European Telecommunications Standards Institute.
ETSI TS 101 903 V1.4.2 (2010-12) Electronic Signatures and Infrastructures (ESI); XML Advanced Electronic Signatures (XAdES)
`https://www.etsi.org/deliver/etsi_ts/101900_101999/101903/01.04.02_60/ts_101903v010402p.pdf`

[12] European Telecommunications Standards Institute.
ETSI TS 102 778-1 V1.1.1 (2009-07) Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 1: PAdES Overview - a framework document for PAdES
`https://www.etsi.org/deliver/etsi_ts/102700_102799/10277801/01.01.01_60/ts_10277801v010101p.pdf`

[13] OASIS Standard
PKCS#11 Cryptographic Token Interface Base Specification Version 2.40.

Edited by Susan Gleeson and Chris Zimman. 14 April 2015.
http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/os/pkcs11-base-v2.40-os.html

[14] *JCommander Documentation*
http://jcommander.org/

[15] *Digital Signature Services (DSS), web page of the project*
https://ec.europa.eu/cefdigital/wiki/pages/viewpage.action?pageId=46992515

[16] *Digital Signature Services (DSS) Documentation - "cook book"*
https://joinup.ec.europa.eu/sd-dss/webapp-demo/doc/dss-documentation.html

[17] *Google Chrome Extension Documentation*
https://developer.chrome.com/extensions