TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

EERO AIRASVIRTA

VERSATILE AUTHENTICATION METHODS FOR WEB APPLICATIONS

Master of Science thesis

Examiner: Prof. Hannu Koivisto
Examiner and topic approved by the
Faculty Council of the Faculty of
Engineering Sciences
on 9th November 2016

ABSTRACT

EERO AIRASVIRTA: Versatile Authentication Methods for Web Applications
Tampere University of Technology
Master of Science thesis, 60 pages
December 2016
Master's Degree Programme in Automation Engineering
Major: Automation Software Engineering
Examiner: Prof. Hannu Koivisto
Keywords: Authentication, Smart card, Web application

Today various modern software applications are implemented as web applications. The applications are running on a web server and only the user interfaces and interactions are transferred over the internet. It is also very common that the applications have limitations that who can use them and therefore an access control system is needed. A usual way to limit the access is to show user a login page and require a correct combination of the username and password. Though this may be the most ordinary way, it is definitely not the most secure way.

When the security requirements for the system are higher, a better solution is needed. Fortunately, there are more secure ways to authenticate users. They include, for example, fingerprint scanning, voice recognition and smart cards. In this thesis we are concentrating on the last one of those: smart cards.

The purpose of this thesis is to investigate possibility of using smart card authentication in web applications. This thesis was written as a part of a customer project that also included a proof of concept system implementation and documentation. The authentication system was implemented as a feature to an existing web application. Objective of the project was to develop an end-to-end demo and to find out what would it need to productize such a system.

In the developed proof of concept authentication system, users were able to log in to the application by using smart cards. In the demo system, not all the features that would be needed for a complete smart card authentication solution were implemented but they were identified and documented. Implementing a complete authentication system would require a full infrastructure for managing the smart cards. With enough time it would be possible to develop such a system.

TIIVISTELMÄ

EERO AIRASVIRTA: Erilaiset tunnistautumismenetelmät web-sovelluksissa
Tampereen teknillinen yliopisto
Diplomityö, 60 sivua
Joulukuu 2016
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Automaation ohjelmistotekniikka
Tarkastajat: Prof. Hannu Koivisto
Avainsanat: tunnistautuminen, älykortti, web-sovellus

Nykyisin monet ohjelmistot toteutetaan web-sovelluksina. Sovelluksia ajetaan web-palvelimilla ja vain käyttöliittymät siirretään verkon yli käyttäjien tietokoneille. On myös erittäin yleistä, että sovelluksissa on rajoituksia siinä, kuka niitä voi käyttää. Tästä syystä tarvitaan erityinen pääsynhallintajärjestelmä. Tavallisesti pääsyä rajoitetaan esittämällä käyttäjille sisäänkirjautumissivu ja vaatimalla oikea käyttäjätunnus–salasana-yhdistelmä. Vaikka tämä on yleisin tapa, se ei kuitenkaan ole turvallisin.

Kun järjestelmän turvallisuusvaatimukset ovat korkeampia, tarvitaan parempi ratkaisu. On kuitenkin olemassa turvallisempia tapoja käyttäjän tunnistamiseksi, kuten sormenjälkiskannaus, puheentunnistus tai älykortit. Tässä työssä keskitytään erityisesti älykorteilla tapahtuvaan tunnistautumiseen.

Tämän työn tarkoituksena oli tutkia mahdollisuutta käyttää älykortteja web-sovelluksiin tunnistautumisessa. Työ kirjoitettiin osana asiakasprojektia, johon kuului myös prototyyppijärjestelmän toteutus ja dokumentointi. Järjestelmä toteutettiin osana toista, olemassa olevaa web-sovellusta. Tarkoituksena oli kehittää toimiva esittelyversio ja selvittää mitä järjestelmän tuotteistaminen vaatisi.

Toteutetussa sovelluksessa käyttäjät pystyivät kirjautumaan järjestelmään käyttämällä älykortteja. Prototyyppitoteutuksessa ei toteutettu kaikkia täydellisessä älykorttijärjestelmässä vaadittavia ominaisuuksia, mutta nämä tarvittavat ominaisuudet tunnistettiin ja dokumentoitiin. Täydellisen älykorttijärjestelmän toteutus vaatisi ympärilleen laajan tukijärjestelmien verkoston. Riittävän ajan kanssa tälläisen järjestelmän toteutus olisi kuitenkin täysin mahdollista.

PREFACE

This master's thesis was written for Valmet as Intopalo's customer project during 2016. I would like to thank Valmet, Markku Tyynelä and Janne Kytölä for giving me this very interesting subject to work with. During the project I have learned tremendously about smart cards, cryptography and digital certificates. In addition, I thank Intopalo for making it possible to write this thesis as a work project. Special thanks also to my thesis instructor, Henry Haverinen, who has helped and guided me throughout this project. I would also like to thank Pekka Laitinen for giving valuable information regarding smart cards and the SCS at the beginning of the project. Also thanks to all my colleagues at Intopalo and Valmet for giving advice whenever needed. Last but not least, I want to thank my family and especially my girlfriend Nancy for supporting and pushing me forward during the project.

Tampere, 22.11.2016

Eero Airasvirta

TABLE OF CONTENTS

LIST OF ABBREVEIATIONS AND SYMBOLS

AES	Advanced Encryption Standard
APDU	Application Protocol Data Unit
API	Application Programming Interface
ATR	Answer to Reset
CA	Certificate Authority
CNG	Cryptography API: Next Generation
CORS	Cross-origin Resource Sharing
CPU	Central Processing Unit
CRL	Certificate Revocation List
CSP	Cryptographic Service Provider
CSR	Certificate Signing Request
DCS	Distributed Control System
DES	Data Encryption Standard
DF	Dedicated File
EEPROM	Electronically Erasable Programmable Read-Only Memory
EF	Elementary File
FID	File Identifier
FINeID	Finnish Electronic Identity
HTTP	Hypertext Transfer Protocol
ITU-T	International Telecommunications Union, Telecommunication Standardization Sector
JSON	JavaScript Object Notation
KSP	Key Storage Provider
MF	Master File
NTP	Network Time Protocol
OS	Operating System
PC/SC	Personal Computer/Smart Card
PKCS	Public Key Cryptography Standard
PKI	Public Key Infrastructure
PRC	Population Register Centre
RAM	Random Access Memory
ROM	Read Only Memory
SCS	Signature Creation Service
SHA	Secure Hash Algorithm
URI	Uniform Resource Identifier

1. INTRODUCTION

Web browsers are used increasingly as platforms for running applications. When a software is implemented as a web application, users can quickly start using it since there is no need for a heavy installation process. Very often applications require that users log in to the system. A very common way to log in is by using a username and password. The username and password authentication, however, is not very secure as people tend to choose weak passwords that can be easily cracked. When there is a need for high security, a better system is needed.

The purpose of this thesis was to study alternative methods for authenticating users and how smart cards especially could be utilized in web applications. As a part of this thesis project we also implemented a proof of concept system for giving advice whether the smart card authentication is a feature that should be used in real products.

In Chapter 2, we start by looking authentication overall and discussing about problems in traditional username and password authentication. We also take a brief look into biometric authentication. In Chapter 3 we focus on modern cryptography, and especially asymmetric encryption and digital signatures as they are essential concepts in smart card authentication. Chapter 4 presents the idea of digital certificates and public key infrastructure. In Chapter 5 we will see what are smart cards, how they work and how they can be used. In Chapter 6 we describe the details of the implemented proof of concept authentication system. Then, in Chapter 7 we concentrate on the security of the system and finally in Chapter 8 we will evaluate the usability of the implemented system.

2. AUTHENTICATION IN WEB APPLICATIONS

Authentication is a process of determining whether someone is who he claims to be. In many systems authentication precedes authorization, which means granting permissions for specific users. For example, when a system contains data that only specific users are allowed to read or modify, the users must first prove their identity for the system. When the system knows who the user is, it can check his permissions and grant or deny access. (Rouse 2015.)

Today many applications require some kind of authentication. Especially, in web applications, which can be accessible from anywhere, authentication is very common. By authenticating users, the applications can be customized for every user. More importantly, the application can be protected from unauthorized access.

Credentials used in authentication can be classified in categories called authentication factors. There are three commonly used factors:

- Knowledge factors
- Possession factors
- Inherence factors

These factors are know as something you know, something you have or something you are. The knowledge factors are something you know. Authenticating using knowledge factor requires knowing and remembering some secret, such as usernames, passwords or PIN codes. Knowledge factors are the most commonly used form of authentication. (Rouse 2015.)

Possession factors are items that the user has. They can be files or hardware devices. For example, security tokens, ID cards or mobile phones can be possession factors.

A physical example of such could be a key to the lock. To be able to open the lock, the user must have the correct key available. (Rouse 2014b.)

Inherence factors, or biometric factors, are something the user is. In other words, they are human body characteristics, such as fingerprints, eye retinas or voice patterns. The advantage of inherence factors is that they are usually quick and easy to use. For example, modern smart phones has fingerprint scanners for unlocking the phone. Unfortunately, these types of authentication factors cannot be easily replaced when they are compromised. (Anderson 2014.)

Authentication can require credentials from one or more aforementioned categories. Single-factor authentication is a common term for authentication that requires credentials from only one category. Similarly, authentication that requires credentials from two or more categories is called two-factor or multi-factor authentication. Since two-factor authentication provides usually better security than one-factor authentication it is also often called strong authentication. (Rouse 2015.)

2.1 Traditional authentication methods

As mentioned earlier, the knowledge factors are the most commonly used group of credentials. Especially, username and password combination is used almost always to authenticate in a web application. The reason why it has become so common, is probably that it does not require any special hardware or software from users and it is very easy to understand how it works. This kind of authentication can be very insecure, though. There is a numerous amount of risks and problems related to passwords.

Perhaps the biggest problem comes up already when creating passwords: When a password is too simple, it can be guessed or cracked easily. On the other hand, if it is too complex, it is hard to remember. If it is hard to remember, it is more likely to be written down on a piece of paper and stored in an easily accessible location. If the paper gets stolen, it does not matter how complex the password is. As there should be a unique password for every service, it is even more difficult to remember all of them. (Baldwin 2012.)

One another problem with passwords or any kind of knowledge factor is related to stealing them. Obviously, when your password gets stolen, you should immediately

change it. The problem is that it is basically impossible to know if it has been stolen. Usually, the first signs of compromised passwords turn out when it is already too late.

Regardless of how complex passwords you create and how carefully you store and use them, they may get stolen. The service providers have to store passwords in their databases. If they have weaknesses in their security measures, which they too often have, hackers may break into their systems and steal the passwords from there. There are frequently news about well-known companies which say that their systems were hacked and several passwords were stolen.

Although the problems related to passwords are widely known, it is still the most used method for authentication. Despite of the problems, this kind of authentication can be enough for many services and applications. However, when there is a need for better security, it is obvious that usernames and passwords alone are not enough.

2.2 Biometric authentication

Compared to passwords, biometric authentication is much less used and rather new method. In 2004 IBM introduced a first laptop with integrated fingerprint scanner (Germain 2004). The user was able to log in to the computer by scanning his fingerprint. Since then, fingerprint scanners have become more common. Today, many modern smartphones have fingerprint scanner, which can be used for unlocking the device or authenticating in certain applications, such as mobile bank. Scanning fingerprints is not the only biometric option. There are also phones that uses facial recognition or iris scanning to authenticate user (Shah 2016).

Biometric authentication differs from password authentication in many ways. A computer can easily compare given password, or its hash, to the one that was saved previously, and then tell whether it was correct or not. With biometrics the situation is different. For example, in case of a facial recognition, the comparing task is not that easy. Your face does not probably look exactly the same that it looked a week ago, which makes it difficult for computers. Although many modern biometric systems are good in this, no perfect system exists. A system falsely rejecting a legitimate user causes only inconvenience, but accepting false identity may have disastrous impact. (Wayman et al. 2005.)

From the user's point of view, fingerprint scanning for instance, may seem very

convenient and secure way to authenticate. Your fingerprints are always with you and they are not something you could forget. It does not matter whether someone sees you putting your finger on scanner when authenticating. No one can steal your fingerprints. However, this is not the case in real world. You can lose your fingers in an accident which would prevent you logging in anymore to a system that uses fingerprint scanning. No one will probably steal your fingertips, but you are leaving your fingerprints everywhere. Someone with moderate skills could use those to create fake copies of your fingertips. In 2013, a hacker created a dummy finger and was able to spoof iPhone 5S fingerprint scanner within 24 hours of its release (Hern 2014). Normally, when your password gets stolen you change it. Fingerprints and other biometrics, however, are something you cannot change. (Johnson 2016.)

Biometrics should not be used to replace passwords in one-factor authentication. Instead, they could be used in multi-factor authentication to provide additional security for other factors. For example, biometrics could replace the usernames in traditional single-factor authentication where users input their usernames and passwords. This way, the biometrics could serve as user's identity and providing a correct password would prove that identity. (Johnson 2016.)

When there is a plan to start using biometric authentication, one thing to consider is the environment. If there is a lot of background noise, voice recognition systems may not be the most suitable option. Or if the users work in an environment where they easily get dirt in their faces or fingers, facial recognition or fingerprint scanning systems may have difficulties to identify users correctly. This kind of challenges are real for example in various kinds of industrial environments. Since the target environment for the subject of this thesis is more or less industrial environment, biometric authentication methods will not be in the main scope of this thesis.

2.3 Smart card authentication

Due to aforementioned problems with passwords and usability challenges of biometrics, the main scope of this thesis will be in possession factors, especially in smart cards. Smart cards fit well in industrial environments since it is very common that the employees have some kind of identification badges anyway. Usually these badges are standard size plastic cards that have employee's name and photo printed on it for visual identification. Those could be easily replaced with smart cards that look the same but contain also employee's digital identity.

Smart cards can be categorized in possession factors, since they are physical objects that you need to have when authenticating. The idea of using smart card in authentication is based on modern asymmetric encryption, public key infrastructure and digital signatures. We will elaborate those topics in the next chapters. Briefly, the digital signature here is the whole baseline of the authentication process.

The Population Register Centre (PRC) in Finland has published a specification that describes method for using digital signatures in web applications. The specification defines a service called Signature Creation Service (SCS). The specification and the SCS were used as starting point at the beginning of this project. We will describe the SCS in detail in the Chapter 6. (Laitinen 2015.)

3. MODERN CRYPTOGRAPHY

Before one can fully understand how public key infrastructure work it is important to know the basics of modern cryptography and data encryption. There are two techniques for encryption: symmetric encryption and asymmetric encryption.

3.1 Symmetric cryptography

Symmetric cryptography is not actually part of public key infrastructure but it is essential in modern cryptography. Therefore it is presented here only briefly.

In symmetric encryption, which is also called shared or secret key encryption, the same key is used in both encryption and decryption (Figure 3.1). Symmetric encryption can be then used if both sender and recipient of the message know the key. Actually, anyone who knows the secret key can decrypt a message that has been encrypted with that key. Therefore a secure way to share secret keys only between authorized people is needed. This is where asymmetric encryption can help. (Microsoft 2007.)

According to RSA Laboratories (2016a) currently the most popular secret key cryptosystem is DES (Data Encryption Standard). Despite the fact that it is vulnerable to brute-force attacks, many applications still rely on it. Due to its vulnerability, DES has been replaced by the AES (Advanced Encryption Standard). (Kelly 2006.)

3.2 Asymmetric cryptography

Asymmetric encryption uses pairs of keys: *public* and *private keys*. That is why asymmetric cryptography is also called *public key cryptography*. In public key cryptography, each party has their own key pair (Milanov 2009). The public key, as the name suggests, is public and it can be shared with anyone. The matching private

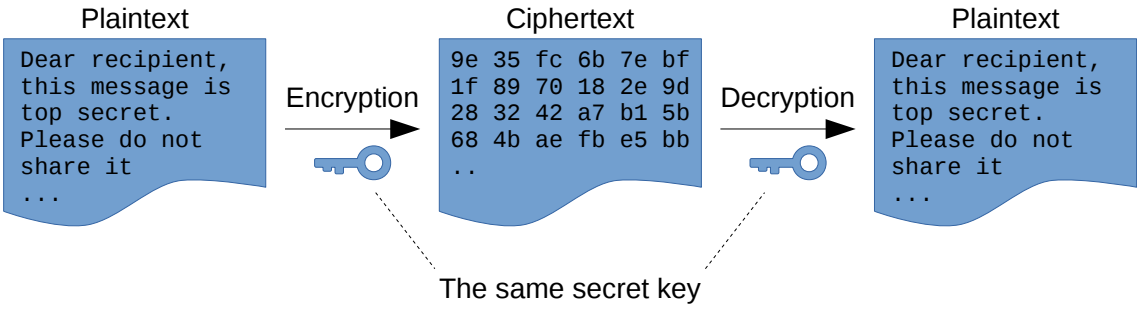

Figure 3.1 *Symmetric encryption. The same key is used in both encryption and decryption.*

key, instead, should be kept secret. One important property of a key pair is that it is very difficult to figure out the private key from the matching public key. Asymmetric cryptography is based on hard mathematical problems where the keys are solutions to. (Bright 2013.)

One of the first publicly available asymmetric algorithms was *RSA*. It was invented in 1977 and named by its inventors Ron Rivest, Adi Shamir and Leonard Adleman. It is built around the problem of prime factorization. One could think that factorization is a simple task, which is actually true when dealing with small numbers. However, when the numbers are big (decimal presentation has more than 300 digits), the task becomes very difficult for even computers. (Bright 2013.)

In addition to RSA, there are other public key algorithms as well. A reason why RSA has become so widely used, is that it implements two important ideas: public key encryption and digital signatures. Both public and private keys can be used for encrypting data. The meaning of the operation depends on the used key. (Milanov 2009.)

In RSA, there are four important properties. If we call the encryption with public key E, decryption with private key D, and a plaintext message M, the following statements are true.

1. Decrypting an encrypted message returns the original message:

$$D(E(M)) = M. \tag{3.1}$$

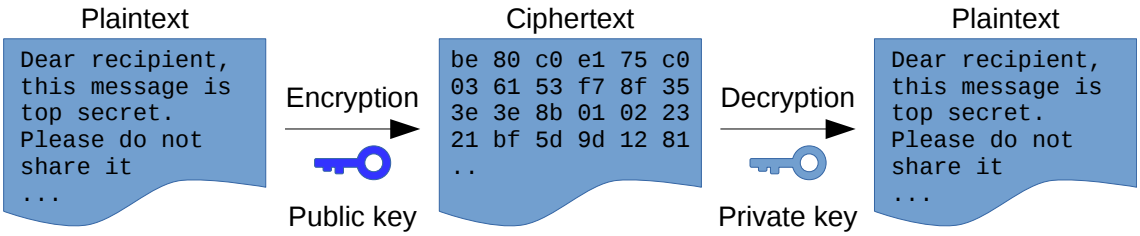

Figure 3.2 *Asymmetric encryption. Public key is used in encryption and private key in decryption.*

2. Calling the operations in reverse order also returns the original message:

$$E(D(M)) = M. \tag{3.2}$$

3. Operations E and D are easy to compute.

4. D cannot be easily figured out from E.

Statement (1) is needed for public key encryption: If a plaintext message is encrypted with public key, it can be decrypted only with its private key. Statement (2) is for digital signatures: If a plaintext message is encrypted with private key, it can be decrypted with its public key. (Milanov 2009.)

3.2.1 Public key encryption

Public key encryption is a mechanism to encrypt a message in a way, that only the holder of the private key can decrypt the message. For example, let's imagine two persons, Alice and Bob. They both have created their own public and private keys. Bob has shared his public key with Alice. Now Alice can use Bob's public key to encrypt a message. Since only Bob knows his private key, only he can decrypt the message (Figure 3.2). Then if he would like to send a message to Alice, he would use her public key for encryption. (Milanov 2009.)

Unlike one might think, public key encryption is not really good for general-purpose encryption. Compared to symmetric encryption, it requires more processing which makes it slower and not so well suitable for large amounts of data. However, small chunks of data can be encrypted rather quickly and therefore it suits well for en-

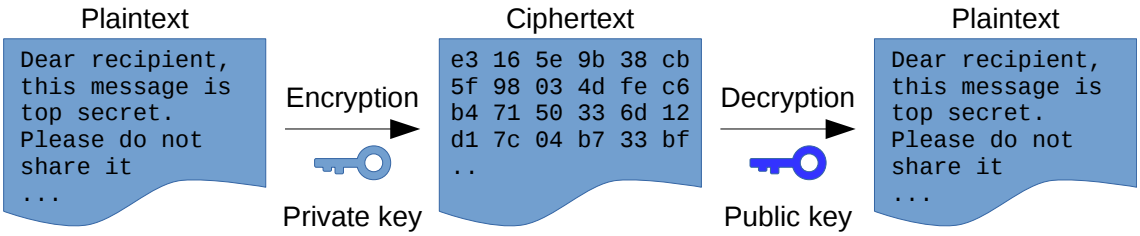

Figure 3.3 *Asymmetric encryption. Private key is used in encryption and public key in decryption.*

crypting keys used in symmetric encryption. For example SSL protocol uses this approach. (Ballard 2012.)

Data can be encrypted also by using the private key. In that case the public key will be used for decryption (Figure 3.3). As the public key is public by definition, anyone could decrypt the data and therefore private keys should not be used for encrypting sensitive data. Nevertheless, this is a very important property of asymmetric encryption as it proves that the data was encrypted by the key owner's private key. When the data is encrypted with a private key, it is called digital signature. (Ballard 2012.)

3.2.2 Digital signatures

Traditionally, a unique hand-written signature in bottom of a letter has been used to prove identity of the sender. A recipient of the letter assumes that the signature is genuine and can be sure that the sender is actually the one who he claims to be. One can easily see that this does not actually prove anything. In digital world there must be a way to sign documents in a way that they can really be trusted.

One important part in digital signatures is *one-way hash* (also known as *message digest*). One-way hash is a fixed-length value that can be generated for any variable-length data by using a specific hash function. An ideal hash function generates unique hashes for the data which means that even a small alteration in data results in a different hash. This property is used for proving integrity of a message. The one-way property of a hash function means that it is not possible to figure out the original data from the hash value. (Ballard 2012.)

Message sender can calculate the hash value of the message and send it together

with the message. Then, the recipient can use the same hash algorithm to calculate the hash of the received message and compare it with the received hash value. If the hashes match, the recipient knows that message is not modified accidentally. However, this does not tell anything about the sender: The recipient cannot be sure if the message was actually from the sender. Hash functions, such as MD5 (Message Digest 5) or SHA-1 (Secure Hash Algorithm 1) are publicly available so anyone can calculate the hashes. (Northcutt 2008.)

If there were a third person between the sender and the recipient and he got the message, he could modify it and calculate a new hash from that and then send those to the original recipient. The recipient would calculate hash from the received (modified) message and see that it matches with the received (modified) hash. He has no ways to detect that the message was modified between him and the original sender. This is where digital signature helps.

As mentioned earlier, public key encryption is not very efficient when encrypting large amounts of data. Since the hashed data is always rather short compared to the original data, it is used for digital signatures. In other words, when one wants to create a digital signature, a one-way hash must be generated first and then the resulting hash can be encrypted by using the private key. The encrypted hash, along with the used hashing algorithm and other possible information is called digital signature. (Ballard 2012.)

When the sender wants to digitally sign a message, he calculates a one-way hash from it and encrypts it using his private key. Then he sends the original message and the digital signature (which consists of the encrypted hash and information about the used algorithm) to the recipient. The recipient then can decrypt the encrypted hash using the public key of the sender. He also calculates the one-way hash from the actual message (by using the algorithm told in the received signature) and finally compares this hash to the decrypted hash (Figure 3.4). If they match, the data has not been changed after it was digitally signed. If they do not match, the data or the signature may be corrupted accidentally or they might have been tampered by someone else.

When using digital signatures, a message recipient can validate message authenticity in addition to message integrity. Only the owner of the private key can generate digital signature for a message. Now, if there were a third person between the sender and the recipient, he could still modify the message, but would be unable to create

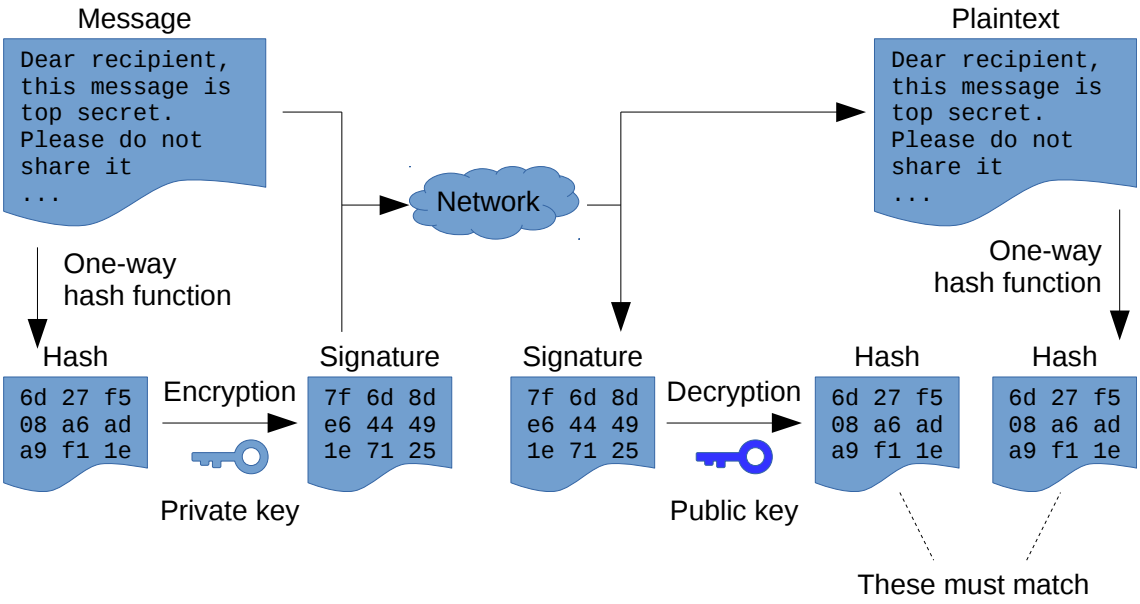

Figure 3.4 *Digital signature.*

a valid signature for that as he does not know the private key of the original sender. This means that if the original receiver gets the message and the valid signature, he can be sure that no one else has modified it before he got it. (RSA Laboratories 2016b.)

One important property that digital signatures provide is called non-repudiation. It means that once one have digitally signed a message, he cannot later deny doing so. Since the private key is supposed to be kept private, only the owner of the key can generate signatures. (Rouse 2008.)

4. PUBLIC KEY INFRASTRUCTURE

In the previous chapter we introduced the most important parts of public key cryptography. There are still questions that need answers: How public keys should be distributed? How can one trust somebody's public key? How to be sure that the public key actually belongs to the person that is claimed?

In this chapter we go through the basic idea of *digital certificates* and the whole system around them. This system is called *public key infrastructure* (PKI).

4.1 PKI overview

When the number of users and applications that uses public key operations is small, managing keys manually might still be a solution. However, when the usage of public keys grows, managing them becomes more difficult. At this point a better approach is needed. (RSA Data Security 1999.)

Keys could be shared through secure directory services, which would provide some protection. However, a possible attacker could pretend to be the secure directory and give users different keys that they believe to get. This would allow the attacker to decrypt and sign messages that were intended for someone else. As public key itself does not contain any information about its owner, there is a need for a secure way to link keys and their owners. One way to bind public key to a person or service is a certificate. (RSA Data Security 1999.)

The main purpose of public key infrastructure is key and certificate management. It enables secure usage of the aforementioned public key encryption and digital signing in large networks. Without PKI, data can still be encrypted and signed, but the identity of the other party cannot be assured. (Rouse 2014a.)

Usually a public key infrastructure consists of both hardware and software components, which are managed by security policies. In addition to digital certificates,

another essential component of PKI is a *certificate authority* (CA). There are other elements in PKI as well but from this thesis point of view, the digital certificates and certificate authorities are the most important ones.

4.2 Certificate authority

In order to work reliably, there is a need for trusted third party in PKI: An entity that users of the system can trust. In PKI this trusted third party is called certificate authority (CA). A certificate authority can be thought as the root of the whole PKI. It is an entity that issues, manages and revokes certificates. As CAs need to be identified and linked with their public keys, they also have certificates. Usually there is more than one CA in a system or context. (Kiran et al. 2002.)

As the CA is the one who issues certificates, it is also responsible of verifying the identity of the entity requesting the certificate. It is up to the CA that how the verification is done. A CA may also delegate this identity verification to a registration authority (RA). In that case, the RA verifies the contents of requested certificate and then provides this information to the CA. (D. R. Kuhn et al. 2001.)

For example, the Population Register Centre (PRC) in Finland issues certificates for citizens in Finland. However, the local police is responsible of identifying the people requesting certificates. Therefore, when requesting a certificate (or actually an ID card containing a certificate), one must go to the police station and prove his identity in a reliable way, for example by showing his passport. Then, if the police can identify the person, they make a certificate request for the PRC which issues the certificate. In this case, the Finnish Population Register Centre is the CA and the local police the RA. (Finnish Population Register Centre 2016.)

Certificate authorities can be divided in two group: Root CAs and intermediate CAs. The main difference between them is that the root CA has issued its own certificate while intermediate CA certificate is issued by another CA. Usually, a root CA issues certificates for one or more intermediate CAs. These intermediate CAs then issues certificates for other intermediate CAs or end users. Hence, CAs often form a hierarchy, where the root CA is the topmost. This hierarchy is also called *chain-of-trust* (Figure 4.1). (Microsoft 2016b.)

Basically, this chain-of-trust means that if a CA is trusted, every certificate it has issued can be trusted as well. Then, by applying this rule recursively, a chain between

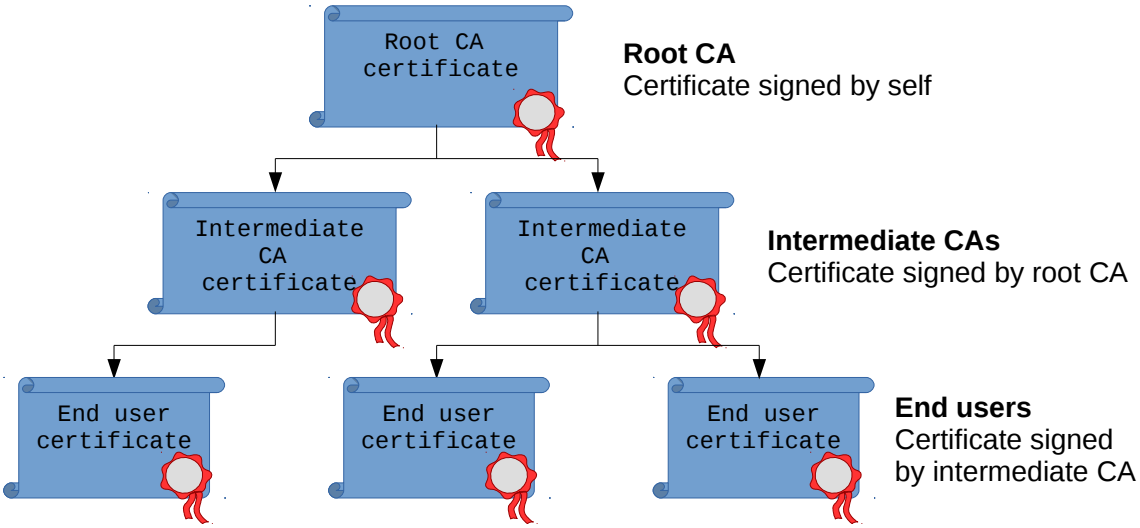

Figure 4.1 *Chain-of-trust.*

the trusted CA and an end user certificate can be formed. Starting points of these chains are called *trust anchors*. Usually, the trust anchor is a root CA that needs to be distributed to relying parties using some other channels than PKI. For example, many web browsers are distributed with several pre-installed root CA certificates. Browser developers have considered these root CAs reliable and have included their certificates to the browser. (Jøsang 2013.)

4.3 Digital certificates

A digital certificate is a document used to identify a person or service and associate that identity with a public key (Ballard 2012). Digital certificates can be compared with passports or driver's licenses. For example, when traveling from a country to another, you go through a passport control where you need to identify yourself. The immigration officer does not know who you are, so he needs some evidence of that. You show your passport to him, who then can see your picture and name from it. Now he knows that according to that passport (certificate) you are the one whose name is in it. How can he then trust that document when he does not know you? The officer sees that the passport has been issued by an authority (CA) that he trusts. Now he knows that the passport is trustworthy and belongs to you. (Posey 2005.)

A digital certificate provide information about the identity of the certificate subject

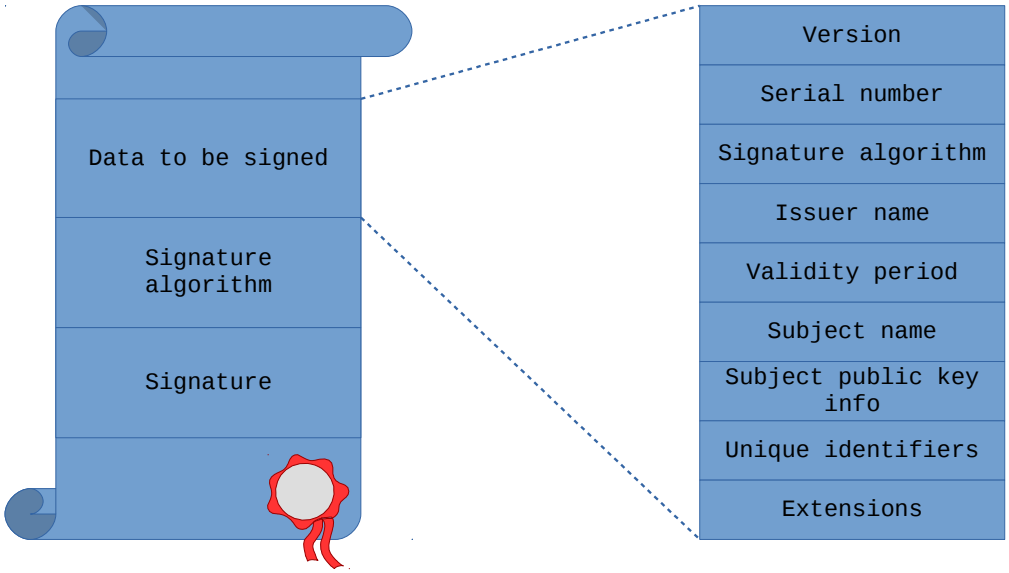

Figure 4.2 *Contents of an X.509v3 certificate.*

and its issuer. Most importantly, it contains the public key of the subject but it may contain other supporting information as well. A widely accepted ITU-T (International Telecommunications Union, Telecommunication Standardization Sector) standard X.509 specifies the format and contents of a digital certificate. (Microsoft 2005.)

An X.509 certificate contains three fields: the actual data that will be signed, the signature algorithm and the signature value. The data field contains information about the subject and issuer, the public key of the subject, validity period and other supporting information. The signature algorithm field contains the identifier of the cryptographic algorithm that the CA have used to sign the certificate. The digital signature of the data field is computed to the signature value field. Contents of an X.509 certificate version 3 is shown in Figure 4.2. The extensions field can be used to add additional information to the certificate. (Cooper 2008.)

The signature is a crucial part of a digital certificate. When a certificate is issued, the issuing CA uses its own private key to sign the data field of the certificate and appends the resulting signature to the certificate. This way the certificate can be validated afterwards by using the public key of the CA. (Microsoft 2005.)

As described in the previous section, certificates form a chain-of-trust. Now this chain can be validated by checking the signature of each certificate. As the CA uses

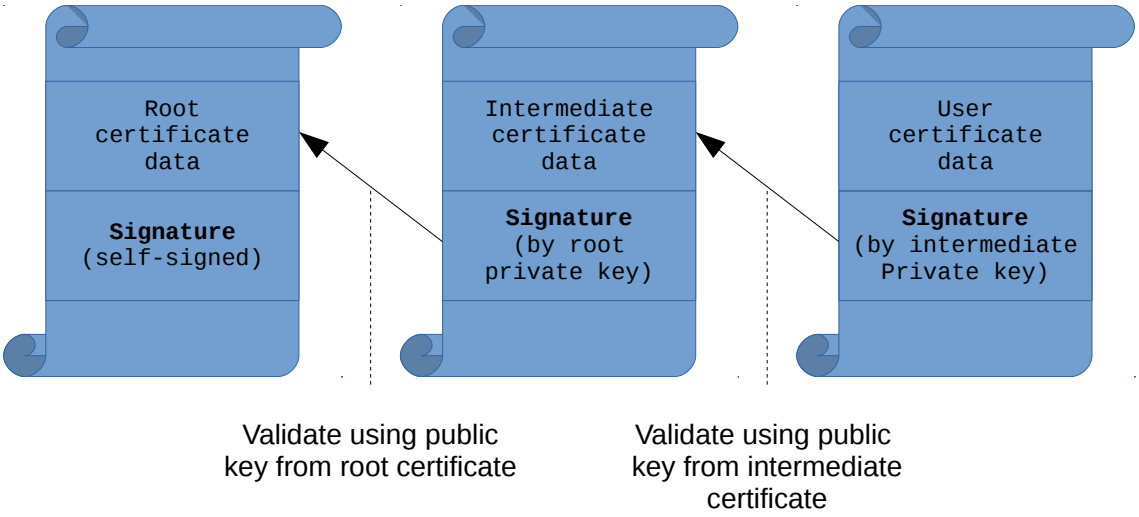

Figure 4.3 *Validation of certificate chain.*

its private key to sign the certificates, its public key can be used to verify them. If we have a pre-distributed trust anchor available (root certificate), we can validate the whole certificate chain (Figure 4.3). If the whole chain can be validated successfully, it means that we can trust the user certificate. (Cooper 2008.)

After validating the signatures in chain, we still need to make sure that the certificates are valid. X.509 certificates contain fields for beginning and ending dates of the validity. If the current date is not between these dates, the certificate should not be trusted. (Cooper 2008.)

4.4 Certificate signing request

Certificate signing request (CSR) is a document that will be sent to certificate authority for applying a certificate. A CSR consists of three fields: the actual certification data, a signature algorithm and a digital signature of the certification data. The certification data field contains the information that the applicant wishes to have in his certificate. The applicant's public key is also included in the certification data. By using his private key, the applicant signs the certification data and appends the signature to the CSR. After this, the CSR will be sent to the CA. (Nystrom and Kaliski 2000.)

When the CA receives a certificate signing request, it first verifies the signature of the request by using the public key from the certification data field. If the request is

valid, the CA can create a certificate with the information from the certification data field. The certification authority uses its private key to sign the certificate. Finally, the CA can send the resulting signed certificate back to the applicant. (Nystrom and Kaliski 2000.)

4.5 Certificate revocation

At some point, there may be a situation when a certificate must be invalidated before its validity ends. For example, if the private key corresponding to the certificate has been stolen or lost, the certificate should be revoked. Certificate authorities need mechanisms to provide status updates for the certificates they have issued. The X.509 standard defines one mechanism for this: *certificate revocation list* (CRL). (D. R. Kuhn et al. 2001.)

A certificate revocation list is a periodically issued signed data structure. CRLs are usually published by the certificate issuer (CA) or a specific CRL issuer authorized by the CA. Revoked certificates are identified in a CRL by their serial numbers. Once a certificate is revoked and added to the CRL, it must not be removed from there before the validity ending date of the certificate is exceeded. (Cooper 2008.)

The base format of an X.509 certificate revocation list is similar to a certificate. A CRL also contains three fields: the actual data that will be signed, a signature algorithm and a signature. The difference is in the contents of the data field. A CRL data field includes, for example, name of the CRL issuer, time and date when the CRL was updated and when the next CRL will be issued and, of course, the list of revoked certificates. The contents of an X.509 certificate revocation list is shown in Figure 4.4 (Cooper 2008.)

When validating certificates, in addition to checking the signatures and validity dates, one must also acquire an up-to-date CRL and check that the certificate serial number is not on that list. Since the update cycle of a CRL depends on the system (can be for example hourly, daily, weekly), the next update field of the CRL should be used to determine if the CRL is up-to-date. Since CRLs are usually not updating in real-time, there is always some delay before the revocation will be noticed. (Cooper 2008.)

There is still one question about CRLs: How to find one? As mentioned in previous section, X.509v3 certificates contain a field for extensions. One commonly used is

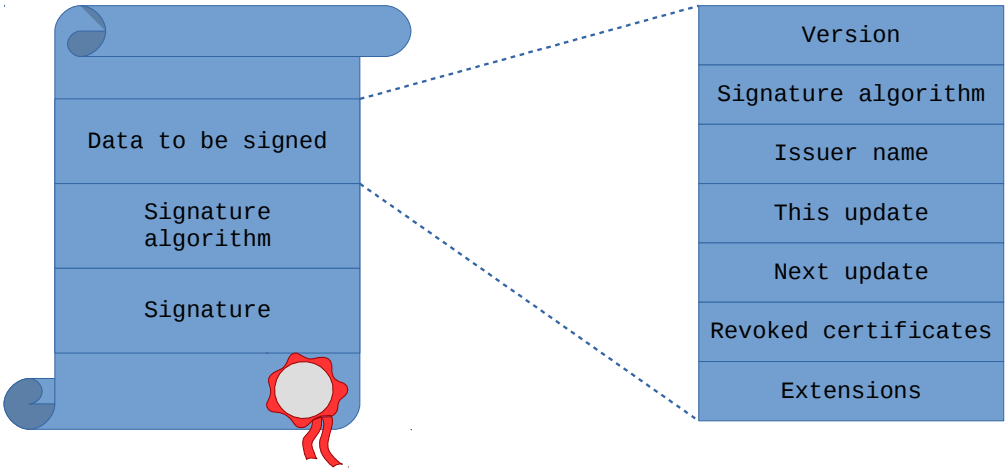

Figure 4.4 *Contents of an X.509 certificate revocation list.*

a CRL distribution points extension. It identifies how the CRL information can be obtained. It may contain, for example, an URI (Uniform Resource Identifier) where the CRL can be downloaded. (Cooper 2008.)

5. SMART CARDS

As described in previous chapters, private and public key pairs are the most essential components in modern PKI. The whole idea of PKI is based on the assumption that the private key is available only for its owner. On the other hand, the owner of the key should have a secure way to store the key without losing it. One place to hold sensitive data securely is a smart card.

5.1 What is a smart card?

Smart cards are often thought as rectangular piece of plastic that has golden connectors on one side. Usually this is the case, since for example modern credit cards are smart cards. Another very common form of smart card can be found inside a mobile phone. SIM (Subscriber Identity Module) cards are also smart cards. Technically, the smart card term is not limited only to the card forms. Smart cards can be for example USB tokens, as well. (Microsoft 2016c.)

A smart card is a small computer that contains an embedded microcontroller and non-volatile memory. These can be used for transmitting, storing and processing data. Modern smart cards even run operating systems and contain file systems that can be used for managing the stored data. One of the most important characteristics of a smart card is that the data stored in its memory can be protected against unauthorized access. This makes it possible to securely save some secret data, such as private keys, to the card. (De Clercq 2016.)

Smart cards can have contacts or they can be contactless. In the former case the data is transmitted by physical connectors. In contactless cards the data is transmitted by an electromagnetic field. Since the credit card sized contact cards are common, we are concentrating on those in this thesis. (Rankl and Effing 2010.)

Since smart cards are used worldwide in everyday life, there are lots of inter-

national standards specifying different properties about them. For example, the ISO/IEC 7816 family defines the fundamental properties and functions of smart cards and the PC/SC (Personal Computer/Smart Card) specifies a cross-platform API (application programming interface) for accessing smart card readers. As always, some of the standards are supported more widely than others and it depends on the card manufacturers which standards they will follow. (Rankl and Effing 2010.)

5.2 Smart card applications

Smart cards have several different applications. In addition to banking (credit and debit cards) and mobile communications (SIM cards), they can be used for example in public transportation for paying tickets or in health care systems for providing personal data about patients. Since the smart cards can be thought as small computers, there are almost unlimited possibilities they could be used for. (Rankl and Effing 2010.)

From this thesis point of view, the most interesting smart card applications are identity and access control cards. These cards can contain digital certificates and corresponding private keys to be used as a part of PKI. One example of such is Finnish electronic identity (FINeID) card. The card is an identity card with card owner's name, photo and other information for visual identification. In addition to that, it is a smart card which can be used as an electronic identity. (Finnish Population Register Centre 2016.)

The FINeID card contains two different user certificates, an authentication and encryption certificate, and a signature certificate. The card also contains private keys matching with the public keys of certificates. The private keys can be used to sign or encrypt data after the user has entered correct PIN code. However, reading the private keys is not possible. In addition to the user certificates, there is also an intermediate CA certificate (VRK Gov. CA for Citizen Qualified Certificates - G2) that has issued the user certificates and a root CA certificate (VRK Gov. Root CA) that has issued the intermediate CA certificate. (Sievänen and Partanen 2004.)

The FINeID card can be used for authenticating securely in some public administration services, such as Kela, the Social Insurance Institution of Finland. In practice, anyone could create a service where one could authenticate himself using the FINeID

card, as long as the root CA certificate (VRK Gov. Root CA) is trusted. The card can be also used for encrypting emails or signing documents. (Finnish Population Register Centre 2016.)

5.3 Smart card security

Smart cards are designed to be tamper resistant. One of the most important advantages of them is that the stored data can be protected against unauthorized access. The data is accessible only through a serial interface that is controlled by the operating system of the card. The operating system can restrict the access to certain parts of the storage in a way that it can never be read, written or erased outside from the card. Therefore it is possible to store confidential data securely in the card. This is why smart cards are highly suitable for PKI. (Rankl and Effing 2010.)

Why should one then store his private keys in a smart card rather than for example on hard drive? Computers are usually connected to the internet, which makes them more vulnerable to different kinds of remote attacks. If the computer gets infected with a malicious software, it may steal the private keys easily. Smart cards instead, are most of the time detached from the card readers, which makes it impossible to use them remotely. Even if the card was connected to the card reader, the operating system of the card would prevent reading restricted data from the card. This way the security-critical private key operations can be isolated from the other parts of the system. (De Clercq 2016.)

One another valuable security feature is that modern smart cards are capable of performing PKI operations, such as encryption and key generation. This means, that a key pair can be generated and the private key can be used for cryptographic operations directly on the card. This way the private key will never need to leave the card or be exposed to the host system. (Kiliçli 2001.)

Even though smart cards are considered secure, it is good to keep in mind that no perfect system exists. Kömmerling and M. Kuhn presented already in 1999 several techniques for extracting protected data from smart cards. Some of them require harming the card by for example removing the chip from the card. More dangerous attacks are ones that do not harm the card, as the card owner might not notice that his private keys has been stolen. If an attacker can steal the card, extract the private keys and return the card imperceptibly, he may be able to abuse the keys

before the matching certificates are revoked. (Kömmerling and M. Kuhn 1999.)

Card manufacturers are developing more security features for smart cards every day. At the same time, crackers are trying to bypass them. This forms an infinite loop where both sides are forced to develop better technology. Fortunately, the security technology is developing faster than cracker methods. Exploiting smart card vulnerabilities also requires usually very high technical expertise and in some cases very expensive equipment. (Bezakova et al. 2000.)

As usual, there are exceptions: In 2010 two university students were able to bypass the security measures of a smart card which was deemed to be one of the most secure on the market. By using only rather cheap equipment, they were able to conduct a man-in-the-middle attack to smart card system that had some flaws. This is a classic example showing that when developing any security system, it will be as secure as its weakest link. (Lee and Pahl 2010.)

5.4 How smart cards work?

A modern smart card consists of a processor (CPU, Central processing unit), an I/O port and three types of memory: RAM (Random Access Memory), ROM (Read Only Memory) and EEPROM (Electronically Erasable Programmable Read-Only Memory). The RAM is volatile and therefore will be erased when the card is removed from the card reader. The CPU uses RAM as working memory. The ROM contains the basic components of the card's operating system. Since the ROM is read-only, no changes can be made after the manufacturer has programmed it. The actual data and program code can be written and read from the EEPROM. Since it is non-volatile, its contents are not lost when the card is removed from the reader. The I/O port is used for serial data transfer. The architecture of a basic smart card can be seen in Figure 5.1. (Rankl and Effing 2010.)

5.4.1 Smart card operating systems

Smart card operating systems (OS) differ significantly from the ones in PCs. The main purpose of a smart card OS is to provide security for program execution and protect access to data. There is a large number of different smart card operating systems, since they are usually designed for a specific need of a card manufacturer.

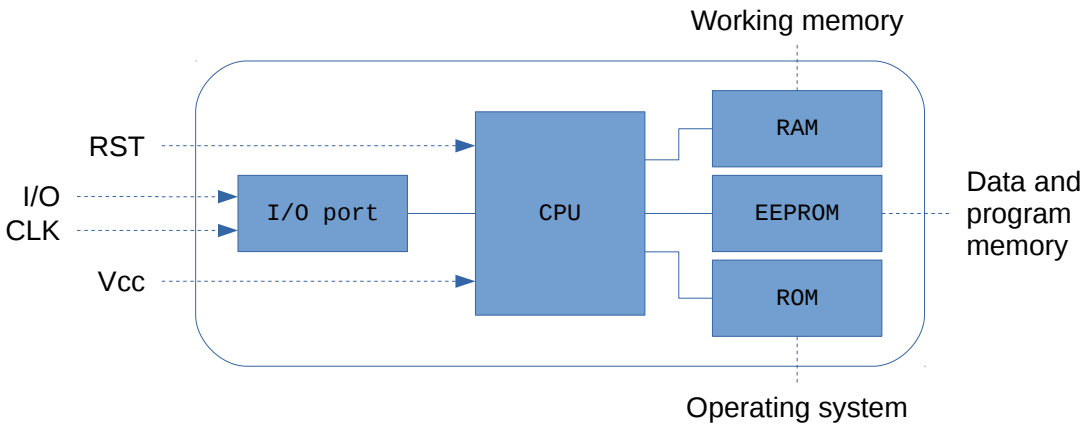

Figure 5.1 *Architecture of a modern smart card.*

Some of them are designed merely for one task, such as access control, while others may support multitasking and provide complex cryptographic functions. Due to severe security requirements and very limited memory capacity it is impossible to produce a general-purpose smart card operating system for everyone's needs. (Rankl and Effing 2010.)

The size of an OS is usually about 10 to 400 KB. Operating systems that are designed to support only one specific application are smaller than multi-application operating systems which support multitasking and running third-party programs. For example, MULTOS and Java Card are smart card operating systems that allow third parties to load their own applications into smart cards. (Rankl and Effing 2010.)

5.4.2 Smart card file system

Most modern smart cards have file systems for storing data. The ISO/IEC 7816 standard specifies the structure of a file system which is similar to the file systems in PCs. There are two types of files: dedicated files (DF) and elementary files (EF). Dedicated files can be thought as directories which can contain more lower-level DFs or EFs. The root directory, which is a special instance of a DF, is called the master file (MF). It represents the entire smart card memory available for storing files. MF, DFs and EFs form a hierarchy which is illustrated in Figure 5.2. (Rankl and Effing 2010.)

Elementary files are used to store the data. They can be classified into working EFs

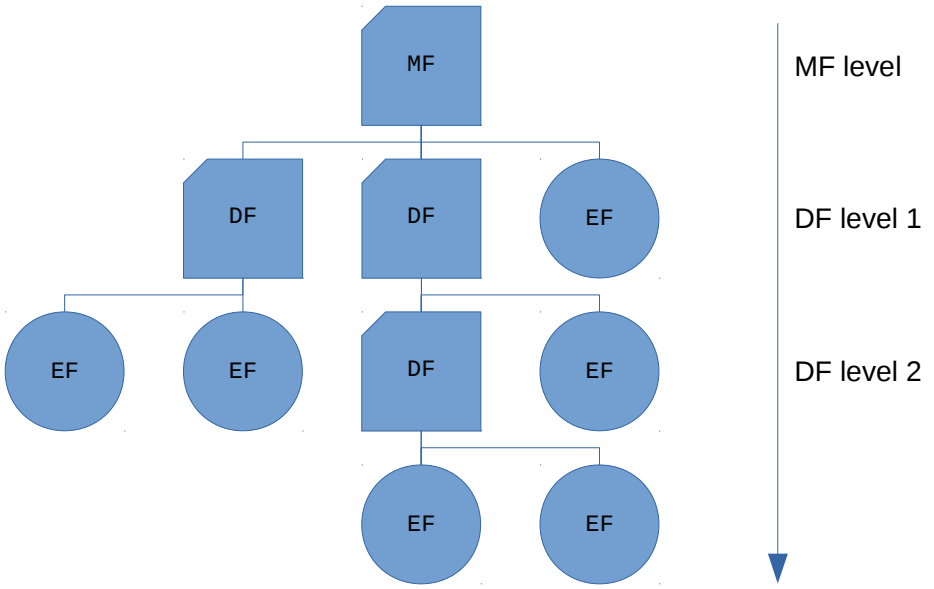

Figure 5.2 *Smart card file system.*

and internal EFs. The former is used for storing data that is intended to be written or read by the smart card reader while the latter is for smart card operating system or program code. The operating system protects the access to the internal files. Every file in a smart card consists of a header and a body. The header contains information about the structure and access conditions of the file while the actual data of a file is stored in the body. (Rankl and Effing 2010.)

There are multiple ways to address files. The ISO/IEC 7816-4 standard defines a file identifier (FID), which is a two bytes long name for the file. Every file has a FID that can be used to select the file. The standard specifies also some FIDs for specific files. For example, the FID of the MF is '3F00'. Another way to address a file is by using application identifier (AID). The AID is assigned by a national or international authority, and each AID can be assigned only once for a particular application. Thus, AIDs can be used worldwide to identify an application. (Rankl and Effing 2010.)

A common convention is to group all files of a specific application into a single DF. This way the file structure becomes clear, and it makes it easy to add new applications to a card. In a single-application smart card, all files can be placed directly in MF or in a single DF below the MF. In a multi-application card, the application files are placed in application DFs below the MF. (Rankl and Effing

2010.)

The Finnish electronic identity card is an example of a single-application card. It contains only a FINeID application. All files that belong in the FINeID application, are located directly below the master file. (Sievänen and Partanen 2004.)

5.4.3 Communicating with smart cards

Smart cards communicate with card readers using half-duplex transmission. This means that the card and the reader must take turns transmitting. While one is transmitting, the other acts as a receiver. The communication is always initiated by the card reader, and the smart card only responds to the commands it receives. In other words, the card and the reader have master–slave relationship where the reader is master and the card is slave. (Rankl and Effing 2010.)

When an ISO/IEC 7816 compliant smart card is inserted into a card reader, the card receives a reset signal, executes a power-on reset and sends an Answer to Reset (ATR) message to the card reader. An ATR is a data string, containing various parameters about the communication protocol and the card. Different cards can be identified from the ATR message they send. After a card reader has received an ATR message and identified the card, it can start communicating with the card. (Rankl and Effing 2010.)

Smart card readers communicate with smart cards by using application protocol data units (APDU). There are two types of APDUs: The card reader sends command APDUs (C-APDU) to the card, and the card replies to them using response APDUs (R-APDU). The structure of APDUs is defined in the ISO/IEC 7816-4 standard. A command APDU consist of a mandatory header and an optional body. Basically, the header contains an instruction and parameters, and the body may contain optional data. A response APDU consist of an optional body and a mandatory trailer. The trailer part contains a return code, which tells if the command was executed successfully or if there were any errors. (Rankl and Effing 2010.)

The ISO/IEC 7816 standard specifies a few basic APDUs. Especially, the part 4 of the standard contains general commands for file management and most of the smart cards understand them. In addition, there are some other more specific APDUs defined in other parts of the standard. For example, the ISO/IEC 7816-8 defines

commands for executing cryptographic functions. It depends on the smart card operating system what APDUs are supported. Some smart cards and operating systems may also support non-standardized proprietary APDUs. (Vandewalle and Vétillard 2000.)

For application developers, using APDUs for smart card communication in their applications would be rather inconvenient. As different smart cards may support slightly different APDUs, there is no universal API that would cover every smart card. In a case where an application is supposed to work only with a specific card (of which available APDUs are known), this might not be a problem, though. However, the abstraction level of APDUs is rather low, which can make it cumbersome to use. (Sauveron 2008.)

These problems were identified already in 1996, when a group of companies formed the PC/SC Workgroup to solve them. In 1997 the working group published the PC/SC specifications (Sauveron 2008). The PC/SC specifies an architecture that encapsulates the functionalities of specific smart cards and defines a high level API for accessing them. In other words, it allows applications to use PC/SC compliant smart cards from different vendors. PC/SC also defines an interface between the card reader and PC. Since smart cards can contain different applications and support different commands, many of those are still used by sending low level APDUs to the card. Therefore the PC/SC should be thought more as an interface between the card reader and PC. (Chirico 2014.)

5.5 Developing applications for smart cards

Smart card applications can mean two different things. They can be either on-card applications or host applications. On-card applications are those that run on smart cards and host applications run on PCs and are used for communicating with smart cards. Due to a wide range of possibilities to use smart cards, there are several types of on-card applications. Some of them are part of card's operating system and are designed for a specific purpose. In those cases, the application cannot be deleted or other applications cannot be installed on the card, but the existing application is supposed to be personalized for individual users. (Cardwerk 2016.)

When one wants to develop own custom on-card application, a multi-application operating system is usually needed. For example, Java Card is a general purpose

OS that supports uploading third party applications on the card. However, it does not automatically mean that if the card has Java Card OS, it would be possible to upload any third party applications there. Card manufacturers may use general purpose operating systems on cards to ease their work for developing applications for specific purpose. After they have finished their application, they can lock the card to prevent users uninstalling it. For example, a Finnish company called Aventra provides smart cards which has Java Card OS and contains a pre-installed PKI application called MyEID (Aventra Ltd 2014). Aventra has locked the card and therefore the application cannot be removed.

When building a smart card system, one should examine existing smart card solutions before starting to develop own custom on-card applications. For example, when there is a need for smart cards to be used in authentication, electronic payments or in any common application, there is probably a smart card with pre-installed application available already.

Host applications run on PCs or other clients and utilize smart cards. When one has developed a custom on-card application, it requires also a custom client application. The client application uses the card by communicating with custom APDUs that the on-card application supports. When the card has a common, perhaps even standardized application, most likely there is a client software or an API already available. From this thesis point of view, the most interesting applications are the ones that can be used in authentication. (Vandewalle and Vétillard 2000.)

In the following subsections, we go through a few technologies that are relevant in terms of smart card authentication. They include both on-card application and host applications or APIs.

5.5.1 PKCS #15 and ISO/IEC 7816-15

In 1991 RSA Laboratories published a series of specifications called Public-Key Cryptographic Standards (PKCS). They have become de facto standards of public-key cryptography. Currently it comprises 15 parts, each of which strongly related in PKI and cryptography. (Landrock and De Cock 2011.)

PKCS #15 is currently the last part of the PKCS specifications. It defines a standard that make it possible for users to use cryptographic tokens, such as smart cards, to

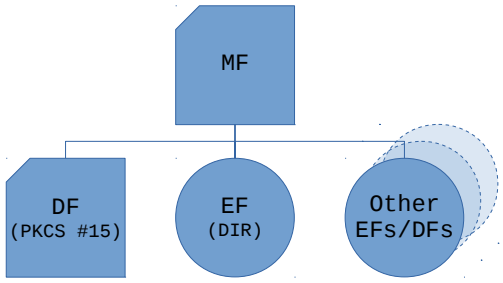

Figure 5.3 *File structure of a card supporting PKCS #15.*

identify themselves to applications regardless of the API the application uses. To achieve this, the PKCS #15 specifies a file and directory format for storing keys, certificates and other security-related information on these tokens. When speaking of PKCS #15 applications, it usually means an on-card application that has structure defined in the PKCS #15. PKCS #15 itself is not an application. (RSA Laboratories 2000.)

ISO has published the ISO/IEC 7816-15 standard, which is based on PKCS #15. The common core parts in both standards are identical, but the ISO standard does not include parts from PKCS #15 that are not related with smart cards. Since the PKCS #15 is freely available while the ISO/IEC 7816-15 is not, we will concentrate on that here. In many cases, these two standards are compatible with each other. (ISO 2004.)

As mentioned earlier, a common convention is to group files of one application in a single application directory (DF). PKCS #15 also follows this convention. In single-application cards the PKCS #15 application files can be placed directly under the MF. In multi-application cards the application files must be placed under PKCS #15 application directory. A typical file structure of an card supporting PKCS #15 can be seen in Figure 5.3. The PKCS #15 application files are placed under a dedicated file called DF(PKCS #15). The EF(DIR) file contains information about applications in the smart card. For example, if a smart card holds multiple PKCS #15 applications, the EF(DIR) file is used to simplify identification and selection of correct application. (RSA Laboratories 2000.)

A common file structure of the PKCS #15 application directory is shown in Figure 5.4. The EF(ODF) is called object directory file. It is a mandatory file that contains pointers to other EFs in the application directory. The EF(ODF) file can

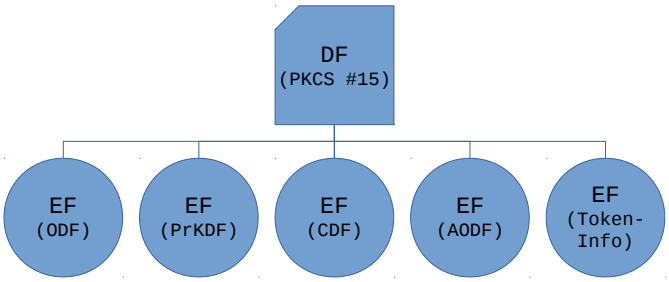

Figure 5.4 *File structure of a PKCS #15 application directory.*

be thought as a root file of the application, since all other EFs of the application can be accessed by using these pointers. Another mandatory file is the EF(TokenInfo). It contains information about the card and its capabilities, such as serial number, and supported file types and cryptographic algorithms. The EF(ODF) and EF(TokenInfo) files can be accessed easily, as their file identifiers are defined in the PKCS #15 standard. (RSA Laboratories 2000.)

The EF(PrKDF) files are private key directory files. They contain general information about private keys, such as labels and identifiers, but most importantly, they contain pointers to the actual keys. If the private keys are protected with a PIN code they also contain pointers to the authentication objects. If the smart card contains public or secret keys, there can also be EF(PuKDF) (public key directory file) and EF(SKDF) (secret key directory file) respectively. (RSA Laboratories 2000.)

The EF(CDF) files are certificate directory files. They are similar with aforementioned key directory files, but they contain general information about certificates. If a certificate contains a public key whose private key is also stored on the card, the certificate and the private key must have the same identifier. Again, the file also contains a pointers to the actual certificate files. (RSA Laboratories 2000.)

For storing information about authentication objects, such as PIN codes or passwords, there are EF(AODF) files (authentication object directory file). They are used to control access to other objects. Information about which authentication object is used to protect a key is stored in the directory file of the key. Like the key and certificate directory files the authentication object directory file also contains a pointer to the actual authentication object. (RSA Laboratories 2000.)

A PKCS #15 application may contain other files as well, but the aforementioned are the most important. An example of file relationships of a PKCS #15 application

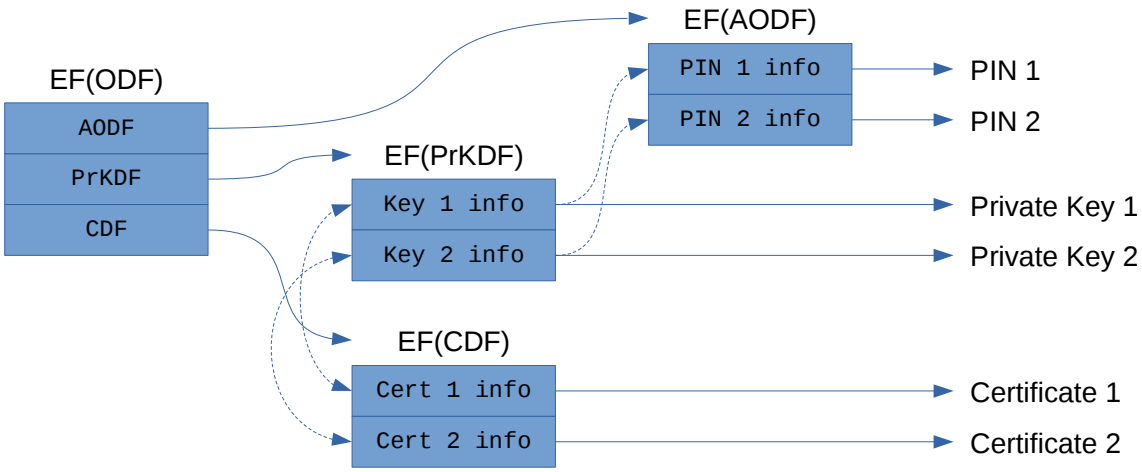

Figure 5.5 *An example of file relationships of a PKCS #15 application.*

can be seen in Figure 5.5. In the example structure, the PKCS #15 application contains two certificates and private keys. Each private key has its own PIN code. (RSA Laboratories 2000.)

The PKCS #15 specifies a standard way for storing certificates, private keys and other data needed for cryptographic operations. Thus, it can be very convenient way to use such application in smart cards that are used for authentication. For example, the FINeID application in Finnish identity cards implements the structure defined by this standard. Since the PKCS #15 specifies where to look these files, it can help application developers to create host applications that work with PKCS #15 compatible smart cards. However, the PKCS #15 application does not solve the issue that different smart cards can support different APDUs.

5.5.2 PKCS #11

Another very important standard related to public-key cryptography is PKCS #11. The standard defines a platform independent API called *Cryptoki*, which is short of cryptographic token interface. The standard specifies the data types and functions that can be used in applications requiring cryptographic services. These are defined using ANSI C programming language. The generic header files are available from the PKCS web page[1], but typically they are provided by the Cryptoki library supplier. (RSA Laboratories 2004.)

[1] `https://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-11-cryptographic-token-interface-standard.htm`

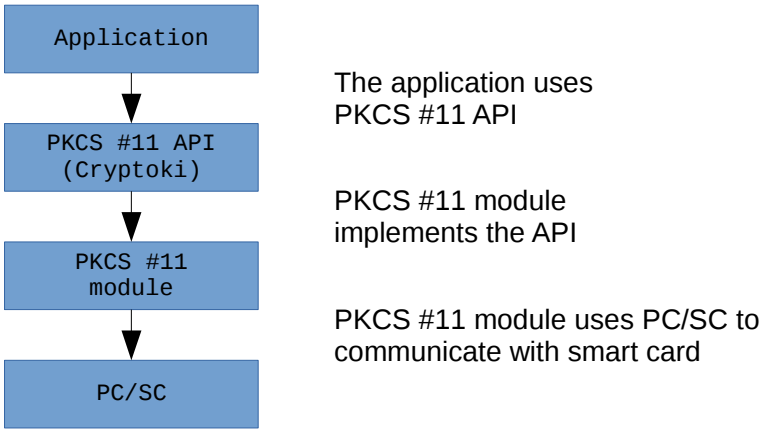

Figure 5.6 *The architecture of an application that uses PKCS #11.*

One purpose of Cryptoki is to isolate a host application from the details of the cryptographic device. In other words, the host application developer does not have to know what kind of smart card the system is using or what are the correct APDUs to be used. Another goal of Cryptoki is resource sharing: A single cryptographic device can be used in several applications. On the other hand, one application can use multiple devices simultaneously. (RSA Laboratories 2004.)

PKCS #11 specifies only the interface, but not the actual implementations. Usually the interface is implemented as a library which is linked to the application. The basic architecture of an application that uses PKCS #11 can be seen in Figure 5.6. Since the Cryptoki API is standardized and different cards support different set of APDUs, the PKCS #11 implementation is responsible for converting Cryptoki functions into correct set of APDUs. Thus, these PKCS #11 implementations depend on the card they support and can be thought as drivers. Usually they are provided by the card vendor and support only specific cards. However, there are also available libraries that support various smart cards. One example of such is OpenSC (see Section 5.5.4). (RSA Laboratories 2004.)

From the software developer's perspective, it is rather easy to create an application that supports smart cards by using PKCS #11. The application does not have to know anything about specific smart cards (or other cryptographic tokens) as long as it uses Cryptoki. However, the Cryptoki has one little drawback. Applications need to be manually configured to use a specific PKCS #11 module. If the application is supposed to support any smart card, the developer cannot know what module an end user will use and therefore the application can not depend on a specific

module. (OpenSC 2014a.)

The Cryptoki API can be used for cryptographic operations, such as data encryption and signing but also in smart card management. In other words, the API can be used in applications for initializing and customizing smart cards for end users. This makes it possible to use the same API in smart card management applications and in end user applications. (RSA Laboratories 2004.)

5.5.3 Microsoft CryptoAPI and Cryptography API: Next Generation

When developing an application for multiple platforms, the PKCS #11 is basically the only option. However, if there is a need for Windows-only application, there is an alternative for PKCS #11. Microsoft has developed an API that provides several cryptographic functions. Formerly, the API was called *Microsoft CryptoAPI*, or *MS-CAPI*. In Windows Vista, Microsoft introduced a long-term replacement for the API called *Cryptography API: Next Generation (CNG)*. (Microsoft 2016a.)

The CNG is an application programming interface that enables developers to add cryptographic features to Windows-based applications. The API provides basic encryption and decryption operations as well as methods for creating and verifying digital signatures. (Microsoft 2002.)

The CryptoAPI and CNG are designed to be easily extensible (Microsoft 2016a). Functions and algorithms that the CryptoAPI provides are implemented as independent software modules called *cryptographic service providers* (CSPs). In CNG these cryptographic service providers were separated to *cryptographic algorithm providers* and *key storage providers* (KSPs). Microsoft have developed some basic providers which are distributed with the operating system. But since the APIs are designed to be extended, third parties can develop their own providers as well. (Microsoft 2016d.)

In practice, the CSPs and KSPs have the same task that PKCS #11 implementations. They all are basically smart card drivers. They are responsible for translating the high level functions of CryptoAPI or CNG to a set of low level APDUs. Smart card vendors need to develop a suitable providers for each of their smart cards. However, a fully functioning cryptographic service provider for a smart card may

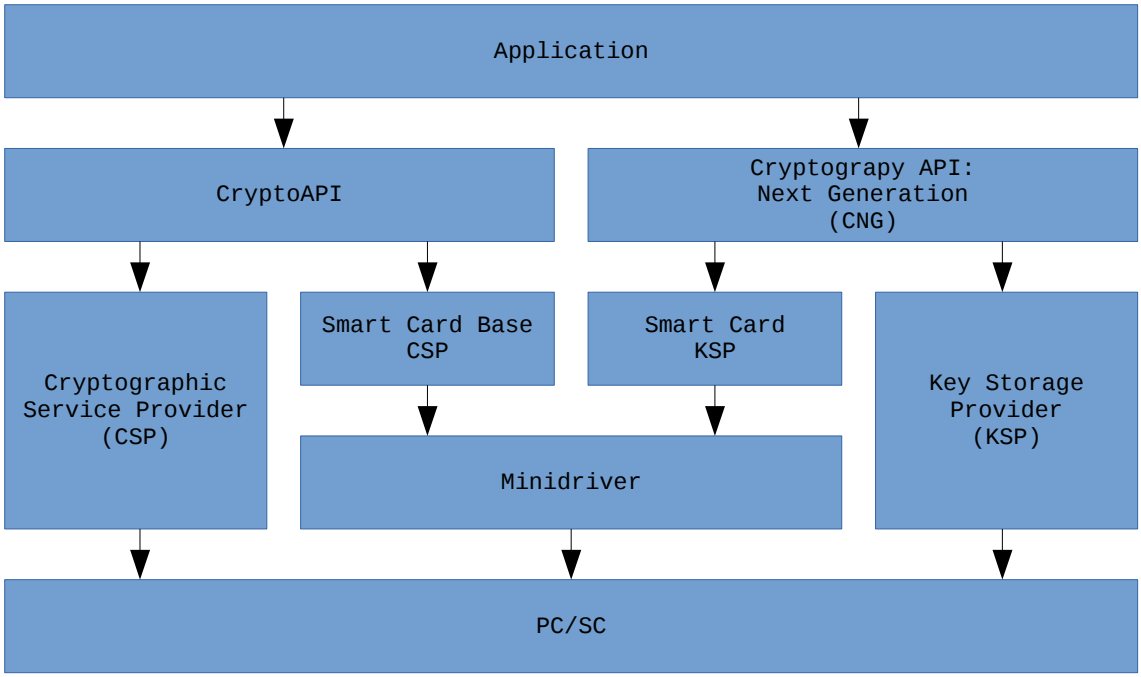

Figure 5.7 *Options for architecture of CryptoAPI or CNG application.*

contain a lot of code, but only a small piece of that is smart card specific. Due to this, Microsoft created a concept called *minidriver*. (Derek 2006.)

The base idea of minidriver is that it implements only the smart card specific parts of the CSP or KSP, which makes things easier for card vendors. Microsoft has developed base providers which contains the common parts of the smart card providers. In the CNG, the provider is called *Smart Card Key Storage Provider* and in CryptoAPI it is called *Smart Card Base Cryptographic Service Provider*. If the smart card vendor has developed a smart card minidriver for Smart Card Base CSP, it is also compatible with Smart Card KSP. The architecture of an application that uses CryptoAPI or CNG can be seen in Figure 5.7. (Microsoft 2016e.)

5.5.4 OpenSC

When smart card vendors develop drivers (PKCS #11 library, full CSP/KSP or minidriver) for their smart cards, they usually support only one specific card. When we are developing a smart card system and we know that our system will use only specific cards from one vendor, we can easily use the driver provided by the vendor. However, if the system should work with cards from different vendors, using a card

specific driver is not really an option. Luckily, there is a solution called OpenSC.

OpenSC is an open source collection of libraries and tools for smart cards. It implements the PKCS #11 API, which means that it can be used with any application that supports the API. Compared to smart card specific PKCS #11 drivers, the OpenSC supports multiple smart cards. Basically, there is an OpenSC driver for every supported card. Although the list of supported smart cards is quite long, naturally there are some exceptions and not every single card is supported. However, this is not a big problem as the OpenSC is open source and anyone can write a new driver for a card. (OpenSC 2016a.)

Smart card drivers provided by the vendors are usually distributed as compiled binary libraries. Unfortunately, card vendors may not necessarily develop the drivers for every operating system or platform. Since the OpenSC is a cross-platform software, it can be used when the card vendor does not provide a proper driver. (OpenSC 2014b.)

OpenSC contains also several command line tools for personalizing, debugging and testing smart cards. They can be used for example for browsing the smart card file system, creating new files or changing PIN codes. (OpenSC 2012b.)

6. PROOF OF CONCEPT SMART CARD AUTHENTICATION

One part of this thesis project was to implement a proof of concept smart card authentication system for an existing web application. In this chapter we will first take a brief look at one example of such an application. After that we will describe the requirements that were set for the authentication and present one way to implement a smart card authentication system. Finally, we go through the details of the system that was actually implemented.

6.1 Valmet DNA

Valmet DNA is an automation system that covers all functions of a distributed control system (DCS). The system is highly scalable from single stand-alone controllers to plant-wide systems. Reliability, availability and security have been taken into account in the system design and architecture. The Valmet DNA consists of multiple products for different functions. With respect to the web applications referred in this thesis, the most interesting product is Valmet DNA report. (Valmet 2016a.)

Valmet DNA Report contains reporting and analyzing tools for operators. It is a web-based portal which makes it very convenient and easily maintainable as the only thing needed to use it is a web browser. The DNA Report supports multiple users, and each user can configure and customize it for their own needs and preferences. To show customized content and data for each user, they need to log in to the system. (Valmet 2016b.)

6.2 System requirements

As this project was an initial research about using smart cards in authentication, the requirements for the system were set somewhat loose. They were mostly set

by the existing system, as we wanted to create new limitations for that as less as possible.

The existing system is a web application and it works on any operating system, therefore we did not want to make any limitations about them in the new system. The smart card authentication should work in any operating system and web browser as well. In addition, the new system should work without browser extensions or plugins.

As the system is a part of an industrial automation system, reliability and availability play an important role. Therefore the system should depend on any external service as less as possible. For example, validating of the credentials should happen internally without depending on external services. Users should be able to log in the system even if there were failures in other parts of the system.

As the users of the system may already use smart cards with some other product or for example as a physical access key, we did not want to make our system such, that it would require of using some specific smart cards. The goal was that if the customer was already using smart cards, this system would also work with them.

6.3 Authentication system

The authentication process on general level is fairly simple. A user navigates to a web page which starts the authentication process. The server behind the page generates a random number, called nonce, and sends it to the user. Then the user digitally signs the nonce using his private key stored in his smart card. The user sends the signed nonce along with his certificate back to the server. Finally, the server verifies the signature using the public key from the certificate and ensures that the certificate is valid and can be trusted. If the signature is valid and the certificate can be trusted, the authentication succeeded and access can be granted for the user.

In a real system, the process is very similar. A sequence diagram of the authentication process and the essential components are shown in Figure 6.1. On the server side there can be seen three main components. The web front end which renders the actual login page, a service for handling sessions (session service), and a service for configuring the users and validating credentials (authentication service). On the

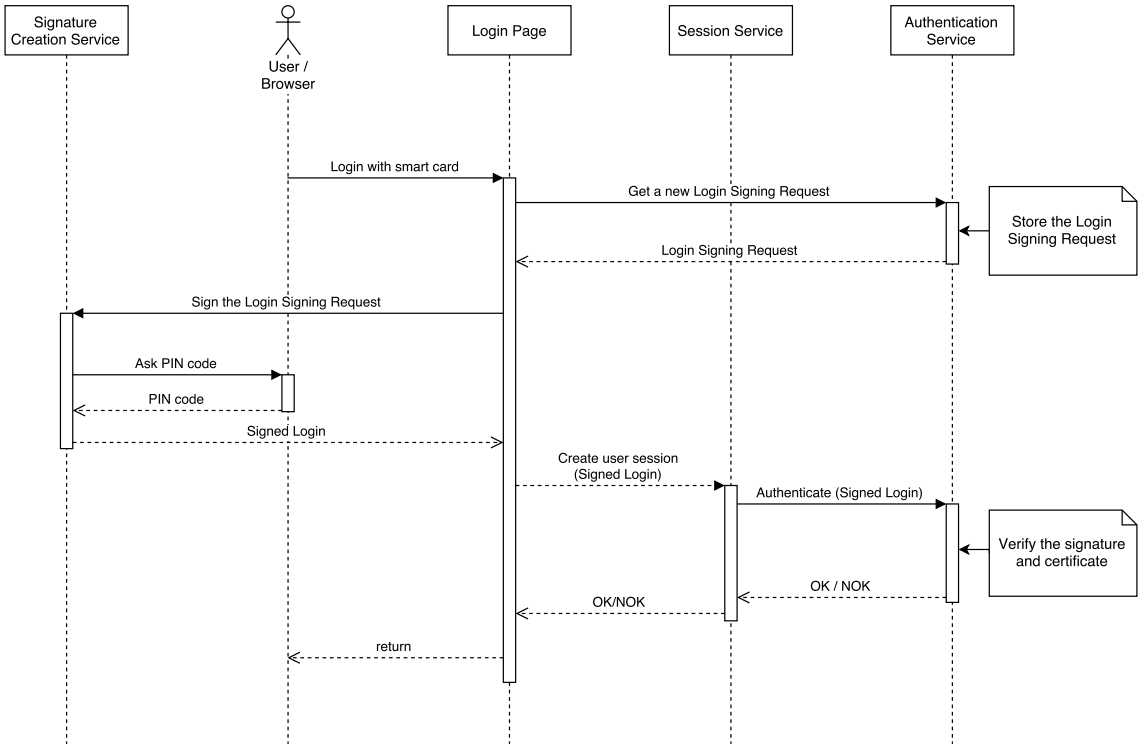

Figure 6.1 *Authentication sequence.*

client side, there is a service for smart card operations (Signature Creation Service, SCS).

At the beginning, the user navigates to the login page. When the page loads, it automatically checks whether the signature creation service is running. If the service is running, the login page shows a button for starting the smart card authentication. When the user clicks the button, the login page sends the authentication service a request for a login signing request. The login signing request is an object containing the nonce value and the ID number of that login signing request. The authentication service creates a login signing request, stores it internally to use later and sends it back to the login page. The login page will then pass the nonce from the signing request to the Signature Creation Service, where the actual signing happens. The technical details of this part of the process will be explained in the Section 6.3.1.

The Signature Creation Service will result a signed login object. The object consists of the digital signature, used hash algorithm and the certificate chain of the user certificate. The Signature Creation Service will send the signed login object back to the login page, which then appends the ID number of the original login signing

request to it.

From here, the rest of the sequence will be similar with the basic username–password authentication. The credentials, either username and password or the aforementioned signed login object will be passed to the session service. The session service will basically forward the credentials to the authentication service which then validates them. This validation process will be described in detail in Section 6.3.2.

The authentication service will then respond to the session service whether the authentication was successful or not. If the authentication succeeded, the session service generates an access token which uniquely identifies the session. The session service stores the token and wraps it inside a web cookie. The cookie will be passed to the login page, which will forward it all the way to the user's browser. Now the user has authenticated successfully and he can access to the system by including the cookie in the page requests. Every request will be routed through the session service, which will check whether the access token in the cookie matches with previously stored sessions.

6.3.1 Generating credentials

As mentioned, the actual signing happens in the Signature Creation Service. The SCS application is a smart card host application running on the user's computer or device. In other words, the same computer where the web browser is running. The question is, how to get the login signing request, or just the nonce, from the web browser to the SCS application.

Essentially, the SCS is an interface and an abstract application for creating digital signatures in web applications. It is defined in a specification published by the Population Register Centre. The basic idea in the SCS is that the smart card host application offers an HTTP (Hypertext Transfer Protocol) interface for requesting digital signatures. In other words, the smart card host application works as a local web server. (Laitinen 2015.)

The SCS interface basically consists of two functions: Version check and signature creation. Parameters for the requests and the response payload are transferred using JSON (JavaScript Object Notation) format. The web application can check the version of the SCS application by sending an HTTP GET request to the SCS

server's request URI (Uniform Resource Identifier) */version*. The response message naturally contains the version of the SCS application but other information as well. For example, supported hashing algorithms are also included in the response. The version check can be used to initially check whether the SCS application is running or not and what are the mechanisms it supports.

The web application can make a signature creation request in two ways. It can send either HTTP GET or HTTP POST request to the SCS server's URI */sign*. However, according to the SCS specification, the HTTP GET method has some limitations with its parameters and therefore the HTTP POST method should be preferred. The only mandatory parameter for the signature creation request is *content*. It contains the actual data to be signed, which in our case is the nonce from the login signing request. Other parameters are optional and they can be used for example to specify the hash algorithm to be used or whether the content is the actual data or already hashed message digest. (Laitinen 2015.)

When the SCS application receives the signature creation request, it reads the user certificates from the smart card currently in the card reader. Then the application shows the available certificates for the user, who then can select the one he will use for creating the signature. When the user selects the certificate he wishes to use, the application requests correct PIN code associated with that certificate. The user enters the code and the application uses it to unlock the private key matching the certificate from the smart card. If the code was correct, the SCS application calls signing function of the smart card application, which then creates the signature. After the SCS application receives the signature, it reads the certificate chain of the user certificate from the card. Finally, the connection to the smart card application is closed and the resulting signature and the certificate chain are wrapped up to a response message.

The response message for the signature creation request contains always at least four parameters: *version*, *status*, *reasonCode* and *reasonText*. Basically, they are used for indicating the version of the SCS application and the status of the signing operation. If the signature creation operation was successful, there will be four other parameters present as well: *signature*, *signatureType*, *signatureAlgorithm* and *chain*. The *signature* parameter contains the actual resulting signature. The *signatureAlgorithm* indicates the algorithm used in signature creation. The *chain* parameter is an array containing the certificate chain of the end user certificate. (Laitinen 2015.)

Finally the web application has received the digital signature, algorithm used in signature creation and the certificate chain. The application creates a signed login object from those and the original ID number of the login signing request and sends it to the session service.

The SCS application clearly consists of two parts: The smart card reader service and the web server. There are some requirements that needs to be taken into account when developing the SCS application.

One requirement for the smart card authentication was that it should work on any operating system. Basically this means, that if we would want to create only one cross-platform implementation of the SCS application, it should use PKCS #11 to communicate with smart cards. If we would create separate applications for different operating systems, we could use CryptoAPI or CNG in the Windows version but in Linux version we would still have to use PKCS #11. Therefore it might be more reasonable to develop only one cross-platform application that uses PKCS #11. There was also one requirement for the smart cards. Our system should not be limited for a specific card type. Therefore, using the OpenSC as a PKCS #11 library would be probably the best option.

For the server side of the SCS application, there is basically only one requirement. It must support *cross-origin resource sharing* (*CORS*). Normally, for security reasons, web browsers restrict *cross-origin* requests that scripts perform. This means, that if the script is downloaded from one domain (origin), it can make requests only to that same domain. However, now in our case, we download the SCS JavaScript module along with the login page from the actual login server but the script makes requests to our local SCS application server which is different domain. CORS is a mechanism that enables these cross-origin requests. (Mozilla Developer Network 2016.)

The main idea in CORS is that when the script from one domain makes a request (HTTP POST in our case) to another, the browser first sends an HTTP OPTIONS request to the latter domain (SCS application server). As a result, the server should respond whether it allows cross-origin requests or not. If the requests are allowed, the browser can continue by sending the actual HTTP POST request to the server. (Mozilla Developer Network 2016.)

6.3.2 Verifying credentials

Verification of the credentials used in smart card authentication consists of several phases. Each of them are important in terms of system security. Each step must succeed in order to successfully authenticate in the system.

When the authentication service receives a signed login object, its job is to verify the signature and certificates from it. The signed login object basically consists of four objects: the ID number of the original login signing request, the digital signature for the nonce, the algorithm used in signature creation and the certificate chain of the user certificate.

The authentication service keeps a list of pending logins. When the user requested a login signing request object from the service at the beginning, the service stored it to the list. Now when the service receives a signed login object, first thing it must do, is to search the signing request matching the ID number from the received signed login object. If the matching signing request cannot be found, the service sends a response that the authentication was failed. If the signing request is found, the service temporarily stores the nonce from it and removes the signing request from the pending logins list.

Next, the authentication service will verify the signature. At the end of the received certificate chain, there is the actual user certificate. The certificate contains the user's public key. The authentication service will use this public key to verify that the received signature matches with the nonce. The service must use here the same algorithm that was used in signature creation. Again, if the signature does not match with the nonce, the service sends a response that the authentication was failed. However, if the signature matches with the nonce, the authentication service knows that at least the private key matching the received user certificate was used in signature creation. Final thing to do is to make sure that the certificates are valid and can be trusted.

For validating certificates, there must be a trusted root certificate (trust anchor) pre-installed in the system. Since we do not want to create any unnecessary dependencies within the system, the root certificate should be part of the authentication service. Now that the service has the root certificate available, it can validate the whole certificate chain in a way that was described in Chapter 4. If the whole chain can be validated successfully, the authentication service can trust the user certificate.

In addition to validating the certificate chain, the authentication service must also make sure that each single certificate in the chain is valid. In other words, it must check that the current time is between the validity start and end dates. Also, the authentication service should regularly fetch an up-to-date certificate revocation list. Each certificate has to be compared with the list. If a certificate can be found from the list, or the certificate validity period has ended (or not yet started), the service sends the response about failed authentication.

The end user certificates must contain some unique data for distinguishing users from each other. Basically, the data can be anything as long as the users configured in authentication service uses the same information. As an example, an email address could be used. The user's email address could be stored in certificate's extensions field, and by using that, the authentication service could search the correct user from its own user configuration. If the matching email address cannot be found, the authentication service would again send the response about the failed authentication. However, if every step this far has succeeded, the service sends the user's configuration to the session service which will then create a new session for the user.

6.4 Smart card creation

Before the smart cards can be used in authentication, they must be personalized for the users. When purchasing smart cards from vendors, they are usually blank cards that must be initialized and personalized before use. Therefore an application for personalizing cards is needed as well.

For initializing and personalizing cards, probably the best option is again the OpenSC. The OpenSC contains several tools which makes the personalization quite easy. Usually, the first step is that we must set up the PKCS #15 file structure. For example, this can be done with a tool called *pkcs15-init*. The same tool can be used also when we set the PIN codes, and generate public and private key pairs. (OpenSC 2012a.)

In addition to OpenSC another very useful toolkit is OpenSSL. It is an open source toolkit and cryptographic library. It contains several tools for example for generating different types of cryptographic keys and certificates. It has also tools for encryption, decryption, data signing and signature verification. Most of the OpenSSL features are available as command line tools and software libraries. (OpenSSL 2016.)

The OpenSC contains also a so-called *OpenSSL engine*. The OpenSSL engines can

be thought as alternative implementations of some cryptographic operation that OpenSSL supports. The engine provided by the OpenSC basically makes it possible to perform some OpenSSL operations in smart cards. (OpenSC 2012a.)

After we have set the PIN codes and created a key pair in the smart card, we still need to create the actual certificate and store it in the card. As described in Chapter 4, we need to create first a signed certificate signing request. Since the signing request has to be signed with the private key and we cannot extract the key from the card, we must create the CSR in the card. This, for example, can be done by using the OpenSSL and the engine provided by the OpenSC. As a result, we get a CSR that is signed with the private key on the card.

Next, we need to create the certificate. Naturally, as we need to form a solid certificate chain from the user certificate up to the system root certificate (the same that we use for validating the certificate chain), there must be an intermediate CA certificate and a private key available for our personalization application. This intermediate CA certificate should be signed directly or indirectly by the root CA. Now, by using the intermediate CA's private key, we can sign the user's certificate, which we can store on the user's smart card. For storing the certificate, we can again use the *pkcs15-init* tool. Now the smart card contains all necessary files and is ready to be used. (OpenSC 2012a.)

6.5 User management

When a certificate is issued for a user, validity dates are used to limit the time period when the certificate is usable. After the validity end date is passed, the certificate cannot be used anymore. However, at some point there may come a situation that the certificate should be invalidated before the validity period is over. There might be several different reasons for this: The user might leave the company or he may have lost his smart card for example. This is where certificate revocation list becomes useful (see Chapter 4).

When a certificate needs to be invalidated, its serial number has to be added to the revocation list. Next time, when the authentication service fetches the list, that certificate cannot be used in authentication anymore. However, if we only keep adding certificates to the list, its size will naturally grow over time. Fortunately, every certificate contain the validity end date. If the certificate has expired, there

is no need to keep it on the CRL.

When a certificate is issued for a user, a copy from it, or at least its serial number and validity dates should be stored in the system that is responsible for CRL management. This way, the system can know when the validity of a certificate ends and it can be removed from the CRL.

6.6 Implemented system

It is clear that implementing a complete system described in the previous sections is a big task. There are several different applications that have to be developed in order to get a product that could be taken into production use. However, the available time for this thesis project was relatively short and therefore the system was implemented only partially. As the main target was to develop a system to show how the authentication can be done by using smart cards, the focus was set to the actual authentication process.

In the beginning of the project, we decided to implement the first version of the system in a way, that it would support Finnish identity cards as smart cards. This way we were able to start by developing the actual authentication system instead of a smart card creation application. Another advantage for starting with the Finnish identity cards was that there was an existing SCS application compatible with them already available.

The application is called *mPollux DigiSign Client* and is aimed for users of Finnish identity cards. It is a smart card host application that implements the SCS specification. The application is commissioned by the Population Register Centre and it can be downloaded freely from the Population Register Centre's website[1]. In addition, Population Register Centre has developed a JavaScript module that provides an easy to use sign function that uses the SCS application. The module is published with MIT license and can be obtained from the SCS web site[2]. The initial version of our proof of concept implementation was developed by using this JavaScript module and the mPollux DigiSign Client as the SCS application.

As mentioned in Section 6.3.1, the SCS application supports checking the version of the application. We used this feature in our system to determine whether the

[1] `https://eevertti.vrk.fi/en/download-card-reader-and-test`
[2] `http://developer.fineid.fi/scs/`

SCS application is running or not. When the user navigates to the login page, it automatically sends a version check request to the SCS application. If the application responds properly, the smart card authentication button is showed to the user. However, if the SCS application does not respond, we can assume that the application is not running and therefore it would be pointless to show the button to the user. In the first version of our demo implementation, the users were still able to log in by using their usernames and passwords.

The authentication service part, which validates the certificates and signature was also implemented only partially. From the security perspective, it would be vital to perform every validation step as described in Section 6.3.2. However, as the first version of the system was developed only for demo purposes, not all steps were implemented. The implemented system checks that there is a login request in the pending logins list that matches with the ID number it receives. In addition to that, the system checks that the signature it receives is valid. When the system is developed further, rest of the steps must be implemented as well.

The implemented system, which used the mPollux DigiSign Client and Finnish identity cards seemed to work rather nicely. However, it is quite obvious that we cannot use them in the end product. Therefore, for the final system, we would need to develop our own SCS implementation that would support smart cards that we want. Since the used smart cards were now provided by the Population Register Centre, there was no need for smart card personalization application. However, when we start using other cards, such an application is needed.

7. SECURITY CONSIDERATIONS

As smart cards are good for storing data securely, it is also important to consider different security aspects of the whole system. In this chapter, we will take a look into the security of the system. We will see what kind of threats there may be and how can we be protected from those.

7.1 Login signing request

In our proof of concept system, the authentication sequence begins with requesting a login signing request from the authentication service. One could quickly question if that is really needed. Why the user cannot create some random data and sign that? The user could send the original data along with the signature, and the authentication service could still verify the signature.

This kind of system would have one notable weakness. The signature and the original data can be reused. The user could simply use always same data for signing, which would obviously result the same signature as well. This, however, is not an issue. The problem is that if an attacker can somehow capture the message containing the signature and the original data, he could also use that to authenticate in the system. Now that the authentication service creates first a nonce that can be used only once, this kind of attacks are not possible. Since the idea of nonce is that it can be used only once, it is important that the authentication service deletes the pending login signing requests after they are used.

7.2 Security of the SCS application

There are a few security related things that need to be taken into account, when implementing or configuring the SCS application. Most importantly, the SCS application should only execute signing request that are needed for authenticating in our

system. As the application is a web server, it has an interface listening to HTTP requests. However, as the service is supposed to be used only from the local machine, we do not want expose that interface for the whole network. Therefore, the SCS application should allow connections only from localhost. (Laitinen 2015.)

Allowing connections only from localhost prevents any other host for requesting signatures directly from the SCS. However, this does not prevent other web sites using similar scripts for requesting signatures. There could be for example a malicious site that contains the same SCS JavaScript module that we have in our web application. If the user then browses to that site, the SCS script could be able to send a signing request to the SCS application. Since the script is run in the user's web browser, the request would be coming from localhost. To prevent this, the SCS application should allow cross-origin requests only from the domain where we are running our own system. (Laitinen 2015.)

7.3 System availability

System availability is an important characteristic of an industrial automation system. Therefore, for example, in our proof of concept we used a locally configured trusted root for certificate validation instead of an external service. If the certificate validation had required some external service and the connection between that and the authentication service had not worked correctly, the users would have not been able to log in the system.

Date and time play an important role in the smart card authentication system. In order to validate the validity period of a certificate, the current time must be configured correctly in the system. By setting a wrong time and date in the system, the authentication service would believe that the certificates are not valid and the users would be unable to log in to the system. Therefore it is crucial that anyone cannot change the configuration. If the system uses for example NTP (Network Time Protocol) to get the time from the internet, it should be ensured that the NTP service provider can be trusted.

If an attacker could be able to somehow change the system clock, in addition to preventing valid users to authenticate, he could be able to log in himself by using an invalid certificate. If the attacker manage to steal a smart card and the matching PIN code, but the card owner notices this, he will add his certificate to the revocation

list. However, since the revocation list contains only certificates that should be otherwise valid, the stolen certificate will be removed from the list when its validity period ends. Now, if the attacker can change the time setting in the system, he can use the stolen smart card for authentication as the certificate is not anymore on the certificate revocation list and the system thinks that the certificate is valid.

7.4 Smart card PIN code

It is possible to initialize smart cards in a way, that they do not have PIN codes. This way, the authentication would be very simple for a user: He only inserts his smart card in a reader, clicks a login button and the rest would happen automatically. After while, the user would be logged in. Although this would be very convenient way for authentication, it would be more insecure.

Initially, the whole point of using smart cards was that it would be more secure than using only usernames and passwords as it would require two authentication factors instead of one. However, if there would be no PIN code, the authentication would be only single-factor as the only thing required would be that the user has access to the smart card. Compared to usernames and passwords, this could be even more insecure solution, since physical smart cards could be stolen more easily. For this reason, there should be usually PIN codes in smart cards. If the system allows users changing their PIN codes by themselves, the possibility of unsetting the codes should be disabled. In a system, where the authentication is used mainly for example for customizing content for different users instead for preventing unauthorized access, using PIN codes may not be totally necessary.

7.5 Multiple certificates on smart card

Usually, different operations in any kind of system requires different access rights. For example, it could be possible to read data without authenticating in the system, while writing requires that user has first logged in. This way the write operations can be recorded in a log file and reading can be very convenient and quick. In a system that uses usernames and passwords in authentication, it could be possible to log in to the system as a read-only user only by entering the username. Write access would then need password as well.

Similar setup could be used in a smart card system as well. It is possible to store several certificates, private keys and PIN codes on a smart card. For example, there could be separate read access and write access certificates and private keys on a card. However, only the private key of the write access certificate would be protected with a PIN code. Now, authenticating with write access would happen in a "normal" way, while authenticating as a read-only user would not require entering PIN code. This could be convenient for users, especially if the write access is needed only seldom. However, the number of user certificates would be then bigger, which could make management of them more difficult.

8. EVALUATION

In this chapter we will briefly evaluate the system under study and the implemented proof of concept. We will also consider how the system could be improved further and what would it need to take it into production use.

8.1 Suitability

A smart card authentication suits well in a web-based automation system. Especially, when there is a need for high level of security, which is often the case in industrial sector, smart cards could be at least one possible solution. Compared to usernames and passwords, the provide much more secure way to authenticate.

Since a smart card and certificate can be thought as digital identities of a user, one card could be used for authenticating in multiple different systems. When there is already one system using smart cards, the existing cards could be easily used in another system as well.

8.2 Usability

The first version of the implemented proof of concept system worked fairly well. However, from the usability perspective, the smart card authentication process was more complicated than the username and password authentication. It required more actions from the user as it was necessary to manually start the authentication sequence by clicking a button on the login page, then select the certificate to be used and finally enter the PIN code. The process also took longer time, since creating the digital signature alone took a few seconds.

There is still room for improving the usability of the system. For example, if the authentication sequence could be started automatically by inserting a smart card to

the reader, instead of requiring user clicking a button first would make the usability better. However, this is something that could be difficult to implement by using similar architecture as we used. As the SCS application is a web server, it only responds to the requests it gets. Therefore the web page could only try to poll the SCS application every now and then to find out whether there is a card in the reader or not.

In this thesis we focused only on contact smart cards. Using contactless smart cards would improve the usability even more. If a user could start the authentication sequence just by lightly touching the card reader with his card, the system would be really convenient to use. However, this can be more difficult to implement, since the OpenSC supports only the contact interface (OpenSC 2016b).

8.3 Deployability

In a system, where users authenticate themselves by using usernames and passwords, the login process is very simple. User enters his username and password to the system, which then probably calculates a hash of the password and compares it to the value stored in a database. When using smart cards, the process is much more complex.

Smart cards require a full infrastructure around them. There must be a system for personalizing cards for users, an application that users need in order to sign data with their smart cards, a service which announces lost or stolen cards and so on. Developing and deploying such a system is very big task that requires a lot of planning, time and resources. Although smart cards provide very good security, it should be thought carefully whether there is actually need for them. In some cases, only updating password policies can improve the security significantly.

9. SUMMARY

Using a web browser as an application platform is very common today. Practically every computer have a web browser installed, which makes it very easy for end users to run web applications on their computers. Many web applications require users authenticating themselves to prevent unauthorized access or to provide user customized content. However, the traditional method to authenticate, by using usernames and passwords, can be very insecure. Users tend to select passwords that are too simple as they do not want to forget them. Fortunately, there are other methods for authentication as well.

In this thesis we focused on smart card authentication. Smart cards are basically small computers that can be used for securely storing data and performing cryptographic operations. What makes smart cards good, is that they are tamper resistant and the stored data can be protected from unauthorized access. Therefore smart cards are excellent for storing for example private keys that are used for encrypting data and creating digital signatures. A user can reliably prove his digital identity by creating a digital signature with his private key. Thus, digital signature can be thought as a foundation for the authentication.

As a part of this thesis project, we developed a smart card authentication system for an existing web application. One key part of the system was a smart card application which was used for creating digital signatures in smart cards. The application worked also as a local web server and it could be accessed through its HTTP interface. When a user was authenticating to the target system, the login page sent a signing request to the smart card application. The application requested a PIN code from the user and used the connected smart card to create a digital signature. The resulting signature was responded back to the target system along with the user certificates from the card. By using the signature and certificates, the target system was able to reliably identify the user.

This kind of system provides very good security compared to traditional usernames

and passwords. However, this also adds complexity to the system architecture, which means that planning and implementing it will take more time. Using smart cards requires large infrastructure around them so it might not be the best idea to implement one unless there is a really need for them. However, when there is a need for very secure authentication, smart cards could be one solution.

BIBLIOGRAPHY

Anderson, M. (2014). *What is Multi-Factor Authentication?* URL: http://www.markandersononline.com/blog/what-multi-factor-authentication/ (visited on Aug. 30, 2016).

Aventra Ltd (2014). *Active Security MyEID Cards*. Tech. rep. URL: https://services.aventra.fi/pdf/ActiveSecurity%20MyEID%20Cards%20white%20paper%20(2p)%20EN.pdf (visited on Aug. 10, 2016).

Baldwin, H. (2012). "Passwords are the weak link in IT security". In: *Computerworld*. URL: http://www.computerworld.com/article/2493184/security0/passwords-are-the-weak-link-in-it-security.html (visited on Sept. 1, 2016).

Ballard, E. D. (2012). *Red Hat Certificate System 8.1. Deployment, Planning, and Installation*. Tech. rep. Red Hat, Inc., pp. 18–20. URL: https://access.redhat.com/documentation/en-US/Red_Hat_Certificate_System/8.1/pdf/Deploy_and_Install_Guide/Red_Hat_Certificate_System-8.1-Deploy_and_Install_Guide-en-US.pdf (visited on July 7, 2016).

Bezakova, I., Pashko, O., and Suredran, D. (2000). *Smart Card Technology and Security*. URL: http://people.cs.uchicago.edu/~dinoj/smartcard/security.html (visited on July 29, 2016).

Bright, P. (2013). "Locking the bad guys out with asymmetric encryption". In: *Ars Technica*. URL: http://arstechnica.com/security/2013/02/lock-robster-keeping-the-bad-guys-out-with-asymmetric-encryption/ (visited on June 30, 2016).

Cardwerk (2016). *Smart Card Operating System*. URL: http://www.cardwerk.com/smartcards/smartcard_operatingsystems.aspx (visited on Aug. 10, 2016).

Chirico, U. (2014). *Smart Card Progarmming. A comprehensive guide to smart card programming in C/C++, Java, C#, VB.NET*.

Cooper, D. (2008). *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. RFC Editor. URL: http://rfc-editor.org/rfc/rfc5280.txt (visited on July 19, 2016).

De Clercq, J. (2016). *Smart Cards*. URL: https://technet.microsoft.com/en-us/library/dd277362.aspx (visited on July 26, 2016).

Derek, A. (2006). *Enterprise Smart Card Deployment in the Microsoft Windows Smart Card Framework*. URL: http://www.giustizia.umbria.it/giustiziapg/resources/cms/documents/WSCFDepl.doc (visited on Oct. 19, 2016).

Finnish Population Register Centre (2016). *Certificates - FINeID*. URL: https://eevertti.vrk.fi/en/certificates-fineid (visited on July 15, 2016).

Germain, J. M. (2004). "IBM Introducing Fingerprint Reader into Laptop". In: *TechNewsWorld*. URL: http://www.technewsworld.com/story/37017.html (visited on Sept. 10, 2016).

Hern, A. (2014). "Hacker fakes German minister's fingerprints using photos of her hands". In: *The Guardian*. URL: https://www.theguardian.com/technology/2014/dec/30/hacker-fakes-german-ministers-fingerprints-using-photos-of-her-hands (visited on Sept. 17, 2016).

ISO (2004). *ISO/IEC 7816-15:2004(en): Identification cards — Integrated circuit cards — Part 15: Cryptographic information application*. Standard. International Organization for Standardization. URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:7816:-15:ed-1:v1:en (visited on Aug. 12, 2016).

Johnson, K. (2016). *5 Things to Consider Before Using Biometric Authentication*. GlobalSign. URL: https://www.globalsign.com/en/blog/biometric-authentication-considerations/ (visited on Sept. 17, 2016).

Jøsang, A. (2013). "PKI Trust models". In: Elçi, A., Pieprzyk, J., Chefranov, A. G., Orgun, M. A., Wang, H., and Shankaran, R. *Theory and Practice of Cryptography Solutions for Secure Information Systems*. IGI Global, pp. 279–301. DOI: 10.4018/978-1-4666-4030-6.ch012. URL: http://folk.uio.no/josang/papers/Jos2013-CRYPSIS.pdf (visited on July 18, 2016).

Kelly, S. G. (2006). *Security Implications of Using the Data Encryption Standard (DES)*. RFC 4772. RFC Editor. URL: http://rfc-editor.org/rfc/rfc4772.txt (visited on June 30, 2016).

Kiliçli, T. (2001). *The Relation of Smart Cards with PKI*. URL: http://www.tldp.org/HOWTO/Smart-Card-HOWTO/smartpki.html (visited on July 28, 2016).

Kiran, S., Lareau, P., and Lloyd, S. (2002). *PKI Basics - A Technical Perspective*. Tech. rep. URL: http://www.oasis-pki.org/pdfs/PKI_Basics-A_technical_perspective.pdf (visited on July 15, 2016).

Kömmerling, O. and Kuhn, M. (1999). "Design Principles for Tamper-resistant Smartcard Processors". In: *Proceedings of the USENIX Workshop on Smartcard Technology*. Chicago, Illinois, USA: USENIX Association, pp. 9–20. ISBN: 1-880446-

34-0. URL: https://www.usenix.org/legacy/events/smartcard99/full_papers/kommerling/kommerling.pdf (visited on July 29, 2016).

Kuhn, D. R., Hu, C. T., Polk, W. T., and Chang, S.-J. H. (2001). *SP 800-32. Introduction to Public Key Technology and the Federal PKI Infrastructure*. Tech. rep. URL: http://csrc.nist.gov/publications/nistpubs/800-32/sp800-32.pdf (visited on July 12, 2016).

Laitinen, P. (2015). *HTML5 and Digital Signatures. Signature Creation Service*. Version 1.0.1. Finnish Population Register Centre. URL: https://eevertti.vrk.fi/documents/2634109/2858578/SCS-signatures_v1.0.1.pdf/e53a601f-a646-4de5-8857-3e5c31d1a9df (visited on Sept. 22, 2016).

Landrock, P. and De Cock, D. (2011). "PKCS". In: *Encyclopedia of Cryptography and Security*. Ed. by H. C. A. van Tilborg and S. Jajodia. Boston, MA: Springer US, pp. 934–935. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_298. URL: http://dx.doi.org/10.1007/978-1-4419-5906-5_298 (visited on Aug. 12, 2016).

Lee, J. and Pahl, N. (2010). *Bypassing Smart-card Authentication and Blocking Debiting: Vulnerabilities in Atmel Cryptomemory-based Stored-value Systems*. Defcon 18. URL: https://www.youtube.com/watch?v=LaBlwTqfayM (visited on July 29, 2016).

Microsoft (2016a). *Cryptography API: Next Generation*. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/aa376210(v=vs.85).aspx (visited on Sept. 27, 2016).

– (2016b). *Securing PKI: Planning a CA Hierarchy*. URL: https://technet.microsoft.com/en-us/library/dn786436(v=ws.11).aspx (visited on July 17, 2016).

– (2016c). *Smart Card Concepts*. URL: https://technet.microsoft.com/en-us/library/dd277376.aspx (visited on July 22, 2016).

– (2016d). *Understanding Cryptographic Providers*. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/bb931380(v=vs.85).aspx (visited on Oct. 19, 2016).

– (2002). *The Smart Card Cryptographic Service Provider Cookbook*. URL: https://msdn.microsoft.com/en-us/library/ms953432.aspx (visited on Sept. 27, 2016).

– (2005). *Understanding Digital Certificates*. URL: https://technet.microsoft.com/en-us/library/bb123848(v=exchg.65).aspx (visited on July 13, 2016).

– (2007). *Description of Symmetric and Asymmetric Encryption*. Version 1.3. URL: https://support.microsoft.com/en-us/kb/246071 (visited on June 29, 2016).
– (2016e). *Smart Card Minidrivers*. URL: https://msdn.microsoft.com/en-us/windows/hardware/drivers/smartcard/smart-card-minidrivers (visited on Oct. 19, 2016).

Milanov, E. (2009). *The RSA algorithm*. Tech. rep. Department of Mathematics, Univeristy of Washington. URL: https://www.math.washington.edu/~morrow/336_09/papers/Yevgeny.pdf (visited on July 1, 2016).

Mozilla Developer Network (2016). *HTTP access control (CORS)*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS (visited on Oct. 20, 2016).

Northcutt, S. (2008). *Hash Functions*. Tech. rep. SANS Technology Institute. URL: http://www.sans.edu/research/security-laboratory/article/hash-functions (visited on July 8, 2016).

Nystrom, M. and Kaliski, B. (2000). *PKCS #10: Certification Request Syntax Specification*. RFC 2986. RFC Editor. URL: http://rfc-editor.org/rfc/rfc2986.txt (visited on Oct. 22, 2016).

OpenSC (2012a). *Card personalization*. URL: https://github.com/OpenSC/OpenSC/wiki/Card-personalization (visited on Oct. 21, 2016).
– (2012b). *OpenSC tools*. URL: https://github.com/OpenSC/OpenSC/wiki/OpenSC-tools (visited on Oct. 21, 2016).
– (2014a). *Creating applications with smart card support*. URL: https://github.com/OpenSC/OpenSC/wiki/Creating-applications-with-smart-card-support (visited on Sept. 26, 2016).
– (2014b). *Frequently asked questions*. URL: https://github.com/OpenSC/OpenSC/wiki/Frequently-Asked-Questions (visited on Oct. 20, 2016).
– (2016a). *OpenSC – tools and libraries for smart cards*. URL: https://github.com/OpenSC/OpenSC/wiki (visited on Oct. 20, 2016).
– (2016b). *Supported hardware (smart cards and USB tokens)*. URL: https://github.com/OpenSC/OpenSC/wiki/Supported-hardware-%28smart-cards-and-USB-tokens%29 (visited on Oct. 25, 2016).

OpenSSL (2016). *OpenSSL – Cryptography and SSL/TLS Toolkit*. URL: https://www.openssl.org/ (visited on Oct. 22, 2016).

Posey, B. (2005). *A beginner's guide to Public Key Infrastructure*. URL: http://www.techrepublic.com/article/a-beginners-guide-to-public-key-infrastructure/ (visited on July 13, 2016).

Rankl, W. and Effing, W. (2010). *Smart Card Handbook*. English. Trans. by K. Cox. 4th ed. Wiley. ISBN: 978-0-470-66089-8. URL: http://site.ebrary.com/lib/ttyk/docDetail.action?docID=10388313&ppg=65 (visited on July 21, 2016).

Rouse, M. (2008). *Definition: Nonrepudiation*. SearchSecurity.com. URL: http://searchsecurity.techtarget.com/definition/nonrepudiation (visited on July 11, 2016).

– (2014a). *Definition: PKI (public key infrastructure)*. SearchSecurity.com. URL: http://searchsecurity.techtarget.com/definition/PKI (visited on July 12, 2016).

– (2014b). *Definition: possession factor*. SearchSecurity.com. URL: http://searchsecurity.techtarget.com/definition/possession-factor (visited on Aug. 30, 2016).

– (2015). *Definition: authentication*. SearchSecurity.com. URL: http://searchsecurity.techtarget.com/definition/authentication (visited on Aug. 25, 2016).

RSA Data Security (1999). *Understanding Public Key Infrastructure (PKI). An RSA Data Security White Paper*. Tech. rep. URL: ftp://ftp.rsa.com/pub/pdfs/understanding_pki.pdf (visited on July 12, 2016).

RSA Laboratories (2000). *PKCS #15 v1.1: Cryptographic Token Information Syntax Standard*. Standard. URL: ftp://ftp.cert.dfn.de/pub/pca/docs/PKCS/ftp.rsa.com/pkcs-15/pkcs-15v1_1.pdf (visited on Aug. 12, 2016).

– (2004). *PKCS #11 v2.20: Cryptographic Token Interface Standard*. Standard. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf (visited on Aug. 18, 2016).

– (2016a). *What are some of the more popular techniques in cryptography?* URL: http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/popular-techniques-in-cryptography.htm (visited on June 30, 2016).

– (2016b). *What is a digital signature and what is authentication?* URL: http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/what-is-a-digital-signature-authentication.htm (visited on July 11, 2016).

Sauveron, D. (2008). "Smart Card Reader APIS". In: *Smart Cards, Tokens, Security and Applications*. Ed. by K. Mayes and K. Markantonakis. Boston, MA: Springer US, pp. 277–293. ISBN: 978-0-387-72198-9. DOI: 10.1007/978-0-387-72198-9_12. URL: http://dx.doi.org/10.1007/978-0-387-72198-9_12 (visited on Aug. 4, 2016).

Shah, A. (2016). "Kill a smartphone password with a scan of your eye". In: *Computerworld*. URL: http://www.computerworld.com/article/3102897/security/

kill-a-smartphone-password-with-a-scan-of-your-eye.html (visited on Sept. 10, 2016).

Sievänen, M. and Partanen, A. (2004). *FINEID S4-1 - Implementation Profile 1 for Finnish Electronic ID Card*. Version 2.1A. Finnish Population Register Centre. URL: https://eevertti.vrk.fi/documents/2634109/2858578/FINEID+S4-1+-+FINEID+Implementation+profile+1+for+Finnish+Electronic+ID+Card%2C+v2.1A+S4-1v21A.pdf/c85e63ef-ab50-48d9-b8e5-347dc94f2495 (visited on July 27, 2016).

Valmet (2016a). *Valmet DNA automation system*. URL: http://www.valmet.com/products/automation/valmet-dna-dcs/ (visited on Oct. 1, 2016).

– (2016b). *Valmet DNA Report*. URL: http://www.valmet.com/products/automation/valmet-dna-dcs/valmet-dna-products/operator-tools/valmet-dna-report/ (visited on Oct. 1, 2016).

Vandewalle, J.-J. and Vétillard, E. (2000). "Developing Smart Card-Based Applications Using Java Card". In: *Smart Card Research and Applications: Third International Conference, CARDIS'98, Louvain-la-Neuve, Belgium, September 14-16, 1998. Proceedings*. Ed. by J.-J. Quisquater and B. Schneier. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 105–124. ISBN: 978-3-540-44534-0. DOI: 10.1007/10721064_9. URL: http://dx.doi.org/10.1007/10721064_9 (visited on Aug. 3, 2016).

Wayman, J., Jain, A., Maltoni, D., and Maio, D. (2005). "An Introduction to Biometric Authentication Systems". In: *Biometric Systems: Technology, Design and Performance Evaluation*. Ed. by J. Wayman, A. Jain, D. Maltoni, and D. Maio. London: Springer London, pp. 1–20. ISBN: 978-1-84628-064-1. DOI: 10.1007/1-84628-064-8_1. URL: http://dx.doi.org/10.1007/1-84628-064-8_1 (visited on Sept. 12, 2016).