

UiO : **Department of Informatics**
University of Oslo

OCR-Based Data Authentication for Online Banking

Marius Portaas Haugen
Master's Thesis Autumn 2016



OCR-Based Data Authentication for Online Banking

Marius Portaas Haugen

1st November 2016

Abstract

This Master's project has investigated the potential for Optical Character Recognition (OCR) based data authentication in online banking, for ensuring authenticity of banking transactions.

An OCR-based data authentication scheme as been designed, and implemented as a high fidelity prototype. The prototype consist of an smartphone application developed for the Android platform, as well as a mock-up e-banking web application. This prototype was evaluated through a pilot usability study, in order to investigate the level of user friendliness which it provides. The results from this pilot study indicates that the authentication scheme itself has potential, however the prototype will have to be improved in order to provide a sufficient user experience.

Acknowledgements

This Master's project concludes my time as a student with the University of Oslo – at least for now. While working on this project I have gained a ton of knowledge, and acquired experiences which will be valuable for the rest of my life.

First and foremost I would like to express my sincere gratitude to my supervisors Audun Jøsang and Christian Johansen. Your support, feedback and most of all patience throughout this project have been invaluable.

I also wish to thank the OffPAD group for letting me participate in their work.

Additionally, I would like to thank Colinda and Spartaco as second readers of this text, input valuable constructive criticism and comments. Not all heroes wear capes!

Furthermore, I would also like to express my gratitude towards my friends and family who have been supporting me while working on this project.

Last, but not least, I would like to thank my dearest Inga, my best friend, partner in crime and spooning cal. This would not be possible without you.

Marius Portaas Haugen
University of Oslo
November 2016

Contents

I	Introduction	1
1	Introduction	3
1.1	Motivation and background	3
1.2	Research questions	7
1.3	Scope and limitations	7
1.4	Research methods	8
1.5	Contributions and results	9
1.6	Structure	9
2	Background	11
2.1	Authentication	11
2.1.1	User Authentication	12
2.1.2	System authentication	13
2.1.3	Data authentication	19
2.2	Online banking and data authentication	22
2.2.1	Transaction Authentication Number	23
2.2.2	Indexed Transaction Authentication Codes	23
2.2.3	Indexed TAN with CAPTCHA	23
2.2.4	Mobile TAN	24
2.2.5	TAN Generators	25
2.2.6	photoTAN	26
2.2.7	chipTan	27
2.2.8	BankID	28
3	Technical background	35
3.1	Cryptography	35
3.1.1	Hash functions	35
3.1.2	Symmetric cryptography	36
3.1.3	Asymmetric cryptography	37
3.1.4	The RSA algorithm	40
3.2	Optical Character Recognition	42
II	OCR Based Data Authentication	51
4	Design	53
4.1	Client side	56
4.2	Server side	59

5	Implementation	61
5.1	Web application	62
5.1.1	Database design	62
5.1.2	Back-end & Front-end	64
5.2	Android client	71
5.2.1	Main Activity - MainActivity.java	74
5.2.2	OCR Activity - OCRActivity.java	75
5.2.3	Authenticate Activity - AuthenticateActivity.java	80
5.3	Communication between webserver and smartphone	81
6	Testing	85
6.1	Usability study – Design	85
6.2	Usability study – Pilot study	88
6.3	Pilot study – Results	90
6.3.1	Pilot study – Observations	90
6.3.2	Pilot study – Analysis	92
6.4	Usability study – Suggested improvements	93
III	Conclusions	97
7	Conclusion & future work	99
7.1	Goal fulfilment	99
7.2	Future Work	100
A	Deliverable 5.2A	101
B	GUI Sketches for the Android application	115
C	Informed Consent Form	119
D	Task List	121
E	Questionnaire	123
F	Coding table for questionnaires	127
G	Coded data from the questionnaires	131

List of Figures

1.1	An early sketch of the OffPAD	5
2.1	Google Authenticator user interface	14
2.2	Public Key Infrastructure for certificates	16
2.3	Firefox address bar indicating valid certificate	17
2.4	Chrome warning user that the certificate used is not signed by a trusted CA	18
2.5	HTML phishing technique	19
2.6	CAPTCHA displaying the text: “smwm”	24
2.7	mTAN ceremony	25
2.8	RSA SecurID OTP generator	26
2.9	Caption	27
2.10	Caption	27
2.11	Login alternatives for skatteetaten.no	29
2.12	Illustration of how BankID <i>could</i> be implemented	30
2.13	BankID on mobile splash screen	31
2.14	Code words for BankID on mobile being displayed in browser	31
2.15	User authentication using BankID on mobile	32
2.16	Transaction details in web interface	33
2.17	BankID SAT application – Splash screen	34
2.18	BankID SAT application – Authentication of transaction	34
3.1	Hash function (SHA1) input and output	35
3.2	OCR overview	43
3.3	Results from OCR experiment - Part II	46
3.4	Text displayed with and without 45 °pan	46
3.5	Results from OCR experiment - Part III	48
3.6	Difference in reflection between matte LCD and glossy LCD	48
4.1	Attack scenario	53
4.2	Ceremony using the OffPAD for data authentication in an online banking use case	55
4.3	Prototype design overview	56
4.4	Smartphone with OffPAD secure backcover	57
4.5	OffPAD backcover	58
5.1	Waterfall Model for Software Development	61
5.2	Database design overview	63
5.3	Table and column overview	63

5.4	Web application - Login page	66
5.5	Web application - Functionality when logged in	67
5.6	Web application - User info	67
5.7	Web application - Transaction registration and transaction details	68
5.8	Web application - Transaction overview field	70
5.9	Web application - Authentication history	71
5.10	Application UI sketches - Part I	73
5.11	OffPAD Android application activities	74
5.12	Main Activity UI - I	75
5.13	Built-in Camera Activity UI	76
5.14	Main Activity UI - II	77
5.15	Screenshot - OCR Activity	79
5.16	Authenticate Activity UI	81
5.17	Setup configuration	83
6.1	Participant session setup	86
6.2	Excerpt of coded data	89
6.3	Results from questionnaires - Question 1.4	93
6.4	Results from questionnaires - Part II	94
B.1	Application UI sketches - Part I	115
B.2	Application UI sketches - Part II	116
B.3	Application UI sketches - Part III	116
B.4	Application UI sketches - Part IV	117

List of Tables

2.1	Sequence of messages and actions in the mTAN scenario . .	25
2.2	Possible BankID work flow	30
3.1	Key agreement using Diffie-Hellman key exchange	39
3.2	OCR experiment - Part I	45
3.3	OCR experiment - Part II - Horizontal text rotation	47
4.1	Sequence of messages and actions in the attack scenario . . .	54
4.2	Sequence of messages and actions for data authentication ceremony	55
6.1	Example of transaction defined in task list	86
6.2	Statement from part III of the questionnaire	88

Part I

Introduction

Chapter 1

Introduction

1.1 Motivation and background

The world constantly continues to progress further into the digital age. Now, more than ever before, we let IT (Information Technology) play an increasingly important role in vital aspects of our lives. On one hand, we use the Internet for a variety of harmless things such as watching pictures of cute kittens, or playing computer games, on the other hand, IT is central to how we manage information considered private, such as finance, tax records and medical information. This would generally be a good thing. It would be hard to argue that IT and the Internet do not enable us to handle these aspects of our lives in a more efficient manner. However, where there are people, there is crime. Or more importantly: where there are people *and* value, there is even more crime. It is my guess that criminals have little or no interest in pictures of cute kittens, and even less interest in stealing them. But what happens when things with real value become accessible through the Internet, such as money?

Online banking, or e-banking, has made it possible for us to manage our finances in a fast, efficient and convenient way. Just sitting behind a computer, we can apply for loans, pay our bills and transfer money. This has made online banking hugely popular in Norway. Online banking is an efficient, convenient approach to banking. In 2015, according to SSB (Statistisk Sentralbyrå, also known as Statistics Norway), 89% of the Norwegian population aged between 16 and 79 years used online banking services in some way [30]. This makes online banking the single most popular way to do banking among Norwegians. Similar trends can be seen in other parts of Europe and in the world in general.

“Safe as the Bank of England”, some say. The quote implies that a bank (or in this case especially the Bank of England) is a place with a high level of security, a place one can trust. If one stores anything of value in a bank, one can expect that it is safe, presumably in a large, well protected vault. Whether being a physical bank, or an online, digital banking system, users expect that a bank is secure and their financial values are safe. However, £133.5m was acquired through criminal activity related to online banking fraud in 2015 in the UK alone, an increase of 64%

compared to the year before [8].

Needless to say, security is essential for these online banking systems. We, the customers, and the service provider (in this case, the bank) are depending on these systems to store and manage our financial funds. There is one concern however. As an increasing amount of value is being handled within these systems, the world of crime also devote more energy into breaking these systems for financial gain. This is a natural side effect of the rise of online banking. Why should a criminal risk her life by storming into a bank trying to rob it using heavy weaponry with the risk of being caught and/or die in the attempt, when she could do it lying comfortably on her couch with a laptop in her lap on another continent risking little more than a mouse elbow? In fact, as we will see, highly advanced, malicious computer software such as "Zeus", SpyEye" and "Dyre" pose a serious threat to online banking as they are being used to steal enormous amounts of money from online banking systems. Malicious software targeting traditional computers is a known, widespread issue. In fact, the European Union Agency for Network and Information Security (ENISA) already back in 2012 recommended all banks to *"assume that all of its customers' PCs are infected - and the banks should therefore take protection measures to deal with this"* [22]. Looking at an online banking system in a simplistic way, we could say that such a system consists of two main components: the client platform (the PC) and the server platform (the bank's server). When we cannot trust one of the components inherently, we have to think differently when it comes to securing the system as a whole. It is clear that in order to create a secure system, we need to add additional components as a remedy to the fact that we cannot trust the client platform. Designers of some online banking systems have included the user's smart phone as a component. However, as we know, when we change what we do, so do the threat agents. According to Kaspersky Labs, 53% of the attacks targeting the Android platform they observed between October 2013 and November 2014 were malicious software designed for financial gain, either through SMS fraud or by attacking online banking systems [16]. This underlines the motivation and interest threat agents have for breaking online banking systems.

While the Internet enables us to do banking in a very efficient way, it also enables criminals to commit crime in an equally efficient way. Even better for the criminals, the Internet enables them to commit their crimes while the risk of them being caught is negligible. By using privacy enhancing services such as TOR¹ (The Onion Router project), criminals can operate anonymously on the Internet which causes much trouble for law enforcement trying to pursue them.

Securing an online banking system is an enormous task, and is outside the scope of this thesis. This text will instead focus on data authentication in online banking systems. It will explore and evaluate existing technology which is being used for data authentication today, in addition to describing an alternative implementation of such a system

¹The Onion Router project - <https://www.torproject.org/>

using classical cryptography and consumer grade OCR (Optical Character Recognition) technology proposed by the OffPAD project². Finally, a prototype of this system is implemented and evaluated.

The LUCIDMAN (Local User Centric ID Management) project took place between June 2011 and October 2013. Its goal was to investigate the "security and the usability aspects for client side management of user and service provider identities". Based on LUCIDMANs' work, the OffPAD project was born. The OffPAD project was initiated in order to explore the possibilities of a device which could help users manage their online identities. An OffPAD is a "trusted device to support different forms of authentication that are necessary for trusted interactions (i.e.user authentication, server authentication)". The project consists of partners

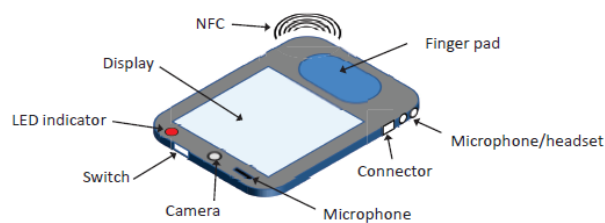


Figure 1.1: An early sketch of the OffPAD

from both academia and industry. TazTag is a French company specialising in developing secure devices, such as tablets and mobile phones. TellU AS, is a system provider and software development company, in addition to Vallvi AS, which conduct business development in the security sector. The GREYC Lab of the E-Payment and Biometrics research unit at ENSICAEN in France, The Security and Usability research unit at the Department of Informatics at the University of Oslo in Norway are both involved.

Based on the previous work of the LUCIDMAN project, and input from OffPAD project partners, a broad range of features has been proposed for the OffPAD. Several features found in the OffPAD could be relevant when securing an online banking system, as it's goal is to facilitate security enhancing operations, including, but not limited to:

- Server authentication
- User authentication
- Data authentication

Originally, the OffPAD project was scheduled for 2013-2016. However, by the time of December 2015 TazTag announced that they were having some financial issues, and were unable to participate in the project as a partner. Since TazTag had such a central role and were to supply the hardware prototypes of the OffPAD, the project as a whole was put on hold

²OffPAD Project - <https://www.offpad.org/>

in January 2016. The project consortium was regrouped and the project was resumed in October 2016.

One of the requested features of the OffPAD is to be able to authenticate data. So what is data anyway? Data can be a broad range of different things: a number, a sentence or even the bit sequence used to represent a picture of a cute kitten. All of this is data. In most cases the term "data" and "information" can be used interchangeably. In an online banking system, there are many different kinds of information (or data) of interest. One example would be the balance of bank accounts. Another example could be the history of transactions tied to bank accounts, or even a transaction itself. A bank transaction is essentially a piece of information which states where you would like to transfer some value from, how much value you would like to transfer, and where you would like to transfer it to. A bank transaction is an important piece of information, both from a user's perspective, as well as from the service provider's. When registering a transaction, the user relies on the bank to carry out that specific transaction according to the particular details specified. If I were to pay an invoice, I would register a transaction corresponding to the details in the invoice in my bank's online system (sometimes referred to as an e-banking system). I then trust that the bank would conduct that particular transaction, according to how I specified it while registering it. If the bank all of a sudden decided to process the transaction using modified or incorrect details compared to those I registered, it would cause me a lot of trouble. In this scenario, my funds could end up in the account of a complete stranger, instead of in the account of my landlord, to whom I owe money for this month's rent.

Banks have a common interest. They would like to trust that the transaction they receive from the user, is what the user actually wants to be processed. If a bank receives a transaction request from a user, specifying that 100,000 NOK is to be transferred from the user's savings account to an account in a Cayman Islands bank, it has to trust that this information is correct, or authentic. If the bank were to process the transaction above, and it turns out that the user actually wanted to transfer 100 NOK to a friend, the bank would find itself in a difficult position. The bank's employees could argue that they just processed the transaction according to how it was specified when it was registered in their system. However, if the user claims that he or she specified a completely different transaction, that would not be fair towards the user. As a result, they would find themselves in a bit of a deadlock.

These examples makes it clear that we have to be able to ensure that the information in the transactions is correct, or more precisely authentic. The information must represent the user's genuine intention.

The OffPAD project wanted to find out if it is possible to ensure authenticity of banking transactions using the OffPAD itself combined with OCR (Optical Character Recognition) technology. This question is the basis for this thesis.

1.2 Research questions

This thesis describes an implementation of a data authentication scheme where the use case is online banking (or e-banking). The main purpose of the system is to provide means to ensure data origin authenticity in a situation where we cannot trust the terminal (i.e., the computer) the user is using for banking activities due to malware infection. In other words, this system must be able to detect and alert unintentional or intentional modification of banking transactions registered within an online banking system. Also, any user of this system should be able to understand what is going on based on the feedback given by the system, so that he or she is able to act accordingly to any occurring events.

As proposed by the OffPAD project, the system should combine traditional cryptography, mobile technology and optical character recognition (OCR). This thesis will investigate different approaches to such a system, and describe a possible design. I will attempt to implement a prototype based on this design, which is to be evaluated in a pilot usability study.

Based on this, the following research questions have been formulated:

1. How can, based on concepts and ideas derived from the OffPAD project, a solution for data authentication of e-banking transactions be designed?
2. Based on this design, how can it practically be implemented in a working prototype?
3. Which level of user friendliness can be achieved with this solution?

1.3 Scope and limitations

This thesis will not describe or discuss how one can design a secure online banking system. Its focus will be exclusively on data authentication, and on how it can be used as a security measure in a online banking system. We will explore a selection of existing technologies for data authentication used today; however this thesis is by no means a complete guide to all data authentication technologies out there. As noted, I will design and implement a prototype of a data authentication scheme proposed by the OffPAD project. This high fidelity prototype will serve as a proof of concept, and is by no means a final product. It has only been developed and tested in a very specific environment, and the results that have been achieved are by no means directly transferable to other environments without further work and modification.

This thesis focus on one specific attack scenario. My primary focus is to ensure data authenticity in a system consisting of two different components which are connected through an insecure channel. The two components are represented as **A**, the user's end point (a PC) and **B**, the service provider's system (server). The insecure communication channel is represented by the Internet. In this scenario, it is assumed that one cannot trust the user's

end point due to malware infection. One assume that there exists a threat agent, whose motivation is to intentionally modify data provided by the user. This modification occurs after the data has been typed into the user's computer, and before the data reaches the service provider's system. More precisely this modification occurs within the web browser, also known as a *Man-in-the-Browser* (MitB) attack. This threat agent is represented as malicious software (malware) which resides in the user's computer. This malicious software is able to modify what is being displayed on the screen as well as modifying all network traffic originating from the machine. It is assumed that the service provider's system is unassailable, and therefore to be trusted in an unconditional manner. There are several other attack scenarios where the use of the approach described in this text would be inadequate.

As noted, this thesis will be using the online banking use case, however the material discussed and the approach described in this text is applicable to a broad range of domains which require data authenticity of visual textual data. Medical prescriptions, medical records and e-voting systems are just a few examples of such domains where ensuring data origin authenticity is essential and our approach could be utilized.

During the work with my thesis I have found that there are technical limitations within the different technologies I have been using which have affected the performance of the prototype. For instance, taking a picture of a glossy computer display introduces too much visual noise to the digital image, which makes it difficult to extract the textual data with satisfactory accuracy. Taking a picture of a less glossy display introduces less interference, which makes extraction of the textual data possible with a much higher accuracy. This is a general problem which applies to any scenario which involves taking a picture of a computer display. Also, image rotation, either horizontally or vertically also affect the performance of the OCR processing. The limitation herein lies in the OCR technology that has been utilized itself, and not in the general concept for data authentication which has been developed. My belief is that with additional improvements and further maturation of these technologies the performance of our system will improve as well.

1.4 Research methods

First, a literature study was conducted in order to understand what data authentication is, and how it applies to modern online banking systems. It was also crucial to understand in what ways these approaches to data authentication are vulnerable, and how threat agents are able to exploit these vulnerabilities, in order to avoid these vulnerabilities in the prototype which was developed.

Based on specifications for a data authentication scheme provided by the OffPAD project, a high fidelity prototype has been implemented in development process utilizing ideas from the iterative development model.

Lastly, a pilot user-based usability study[17, p. 260] was conducted to

evaluate the prototype. In this pilot study, research methods from both qualitative and quantitative research methodologies were utilized, such as observation and questionnaires. The primary motivation behind this pilot study was to identify critical flaws in the design of the prototype itself, as well as any flaws in the design of the study, in preparation for a possible full scale usability test later in course of the OffPAD project. Based on the findings in this pilot study, a set of recommendations has been proposed both for the design of the prototype as well as the research design which should be implemented before further tests are conducted.

1.5 Contributions and results

A demonstration featuring the prototype was held for the members of the OffPAD project in October 2015. The demonstration was well received.

All the code for the prototype has been handed over to the OffPAD project, which plan to integrate the prototype described in this Master's thesis with other, related prototypes developed as part of the OffPAD project.

1.6 Structure

In this text, I will describe an implementation of a data origin authentication scheme. This scheme is based on different technologies ranging from classical cryptography and optical character recognition (OCR), and is applied to a very specific use case. This part, Part I, will describe existing solutions which seek to solve this problem and how they have worked in the past, and are being used by the e-banking industry today. I will also investigate some real life examples of attack scenarios similar to the one we are focusing on. Further, I will introduce some of the technologies and concepts on which our scheme is built.

In Part II I will describe how the prototype was designed, implemented and evaluated. This part also present the results derived from the evaluation.

Part III concludes the the thesis. In this part, I will summarize how the work presented in this thesis relate to the previously defined research questions. I will also propose some ideas for future work related to this Master's project.

Chapter 2

Background

2.1 Authentication

To understand this thesis, an understanding of authentication and related concepts is imperative. I will describe what I mean by the terms "authentication", "authenticity", "integrity", "non-repudiation" and "identification". I will describe how these concepts relate to each other, and how it is possible to prove them using hash functions, symmetric cryptography and asymmetric cryptography. Furthermore, we will also explain the differences between identification (user or system authentication) and data-origin authentication.

Authentication can be defined as the process of confirming the truth of an attribute of a single piece of data (a datum) claimed true by an entity [35]. Something as simple as logging into an online e-mail service would involve two different types of authentication. On one hand, the user (an entity) claims to the service that it is a certain individual, let us say "*Bob123*". In order to confirm towards the server that the user is in fact the individual tied to the account named "*Bob123*", the user enters a password to the service. This password is a shared secret between the individual disposing the named e-mail user account and the service. It is assumed that no one else has knowledge of this password. The service will then check its records, and see if the password entered for the user account "*Bob123*", matches the password entry stored by the service. If the two match, the service will grant the user access to the e-mail account for the individual "*Bob123*".

Imagine the following scenario: our friend Bob has just started going out with Alice, and things are going pretty well for them as a couple. Yet, Alice has a very jealous friend, Eve. Eve does not like the fact that Alice is spending so much time with Bob, and would prefer if Alice and Bob broke off. Since Alice and Bob do most of their communications by e-mail, Eve comes up with a scheme in order to gain access to Bob's e-mail account, which is provided by the fictive e-mail service www.supermail.com. In this way, she can send an e-mail to Alice telling her that the relationship is over, seemingly sent by Bob. To achieve this, Eve plans to set up a web site which looks exactly like www.supermail.com, and trick Bob to visit the site,

and try to log into his account. The site will then redirect Bob to the real www.supermail.com website, while relaying Bob's password to Eve. This is known as a *phishing attack* [21]. To prevent this kind of attack scenario, the service would have to prove towards the user that it is the service it claims to be, so that the user would not be fooled to enter its credentials to a fraudulent phishing site.

If we now assume that Eve has achieved to get hold of Bob's password, she is able to send Alice an e-mail explaining that their relationship is over, which originates from Bob's e-mail address. When receiving the e-mail, Alice could see that the sender address is in fact Bob's e-mail address, but it would be nice to prove that the message in the e-mail in fact is Bob's writing. In other words, we would like some sort of authentication of the message itself. Alice and Bob could, by utilizing relevant technology for e-mail/message authentication such as OpenPGP ¹ achieve this kind of authentication of the messages sent between them. In this scenario, Eve would not be able to spoof an e-mail seemingly sent from Bob without Alice being able to detect it.

As seen above, authentication can refer to much more than just user authentication. I will now move on to explore the different types of authentication more in depth, and some of the technologies used to achieve them.

2.1.1 User Authentication

User authentication is the process where a system verifies and validates a user's identity at the start of an interaction between them. One of the most common ways to achieve this is to use a username and password scheme, as previously described. This authentication scheme relies on two components: a username which identifies the user, and a password which helps the user prove to the service that she really is who she claims to be. The username has to be unique per user, but does not have to be kept a secret. The password however, has to remain a secret shared between the user and the service exclusively. The security of this authentication scheme is based on the fact that the password is kept secret from everyone else but the user and the system. In this way, this scheme only operates with one single secret factor, being the password. This is therefore known as a "single factor authentication scheme". Having only one secret factor creates a single point of failure. If the password is compromised, an attacker would use the password to impost as the real user, and successfully authenticate herself towards the system as the user. In order to make an authentication scheme more robust and secure, one can add one or more additional secret factors. These factors can be divided into the following broad classes:

- Something you know (*e.g. a password*)
- Something you have (*e.g. a one time password generator*)

¹OpenPGP - <http://www.ietf.org/rfc/rfc4880.txt>

- Something you are (*a unique biological attribute, such as a fingerprint or iris*)
- Something you do (*your unique way of doing something, such as walking*)

If one combines two different factors from the classes listed above, one achieves a “two factor authentication scheme”. If more than three classes are combined into a single authentication scheme, this is called a “multi factor authentication scheme”.

For example, through their “2-Step Verification”² technology, Google Inc. allow their users to enhance the security of their accounts by supporting various additional authentication factors for their services. Google 2-Step Verification supports a broad range of different authentication factors such as one-time codes sent as an SMS to the user’s phone, USB security keys and one-time codes using an OTP-system³ [13].

When configured, Google 2-Step Verification will require the user to enter some information obtained using the chosen method in addition to his or her password. In this way, not only one, but two authentication factors are required from the user in order to authenticate. This removes the password as the single point of failure, and a potential attacker will have to have access to both the password and the device/method used to generate the second factor in order to compromise the account.

Google’s one time password implementation is called “Google Authenticator”⁴, and can be seen in figure 2.1. In addition to be used together with a Google account, there are several third party server side implementations which enable the Google Authenticator to be utilized for other types of user authentication such as towards SSH⁵ or VPN⁶ services, as seen in listing 2.1.

Listing 2.1: SSH server asking for a Google Authenticator passcode for user authentication

```
mariusph@safir ~ $ ssh marius@kaffemarius.com
Verification code:
```

This does strengthen the security of the account, and can be further improved by implementing even more factors.

2.1.2 System authentication

System, or entity authentication is defined as the process of verifying the identity claimed by some system entity [9]. Just as the user has to prove its identity to the bank, the bank also has to prove its identity to the user. In terms of online banking, that would mean that the users of an online

²Google 2-Step Verification - <https://www.google.com/landing/2step/index.html>

³A One-Time Password System - <https://tools.ietf.org/html/rfc2289>

⁴Google Authenticator - https://en.wikipedia.org/wiki/Google_Authenticator

⁵SSH - Secure Shell, a secure network protocol, often used to facilitate remote administration of computers using a *nix operating system

⁶VPN - Virtual Private Network, by using secure network protocol for a point-to-point connection one can establish a secure tunnel from one network to another over a potential insecure network

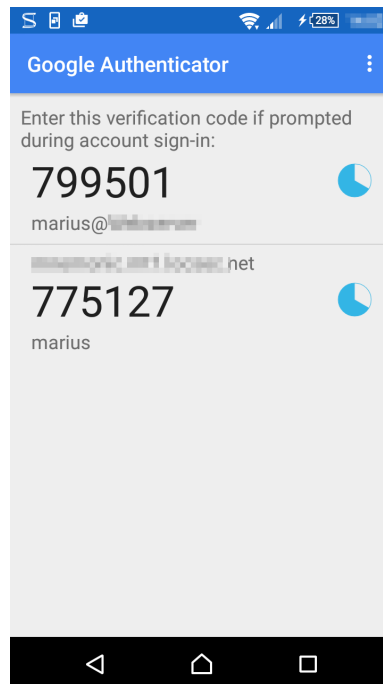


Figure 2.1: Google Authenticator user interface

banking service could verify the identity of the online banking service's website. For websites this is normally achieved by using digital certificates. A digital certificate is essentially just a cryptographic key tied to an identity, for example a person or an organization. The key used in a certificate is one out of two keys in an asymmetric key pair, as described in Section 3.1.3. The other key, the secret key, is kept secret and is only known by the web server. Most certificates used by web servers are based on the **X.509** format [6]. A certificate will include information about, but not limited to [6][p. 16]:

- Version number
- Serial number
- Signature algorithm ID
- Validity period
- Subject name
- Public key
- Certificate signature algorithm
- Certificate signature

The X.509 format is a general certificate format, and can be used for more than just web servers. X.509 certificates used for web servers have the "Extended Key Usage" property set to "*Server Authentication (1.3.6.1.5.5.7.3.1)*", and are called "SSL certificates". Despite this, the

terms “X.509 certificates” and “SSL certificates” are sometimes used interchangeably.

When a browser initiates an HTTPS (HTTP over SSL/TLS) connection towards an HTTPS enabled server, the certificate is sent from the server to the browser. Based on the information found in the certificate and the browsers configuration, it will either proceed to establish the connection, or abort. Since digital certificates can be created by anyone, *anyone* can claim to be *anyone*. One can imagine an attack scenario where an attacker controls the local network (or at least some resources in the local network). The attacker could generate a certificate for a domain, let us say facebook.com and install the certificate on a server under his or her control. By using techniques such as ARP and/or DNS spoofing, the attacker could redirect traffic destined for facebook.com towards this server. This is known as a Man-in-the-middle (**MitM**) [33][p. 257] attack scenario, an attacker can inspect, alter or redirect network traffic between a client and a server. Since the server is HTTPS enabled, and has a certificate for the domain facebook.com the attacker could intercept and decrypt the traffic before sending it towards the real facebook.com servers. This makes it clear that we need some way of knowing whether a certificate is authentic or not, and which certificates we should trust. In order to solve this problem, and for these certificates to have any real value, a global PKI (public key infrastructure) has been established for the type of certificates used by websites.

The global PKI for SSL certificates is structured as a hierarchy of certificate authorities (CAs). At the top of the hierarchy we find a set of what is called “root certificate authorities” (root CAs). The certificates of these root CAs are included in browsers and operating systems, and are trusted by default. On the next level, we find “intermediate certificate authorities” (CAs). These CAs prove their identity as an organization to the root CAs, and get their certificates signed by the root CAs certificates. The intermediate CAs can then use their certificate to sign other entities’ certificates, as long as they prove their identity. When a browser loads a website which presents a certificate, the browser can know if the certificate is authentic or not by validating the certificate’s signature. This is illustrated in figure 2.2.

If a banking institution wants to use an SSL certificate to prove its identity to the user, it generates a certificate request. This request contains information about the organization, such as name and location, the domain it would like to use the certificate for and a cryptographic public key. This request is then sent to a CA which does the job of verifying that the entity applying for the certificate really owns the domain name it wants to use in the certificate. Then, and only then, the certificate is signed using the intermediate CAs certificate, and sent back to the entity which applied for the certificate. The signed certificate is then installed on the organization’s web server, and is used in all HTTPS traffic between the web server and visitors. In this way, the SSL certificate helps users verify that the website they are visiting is authentic.

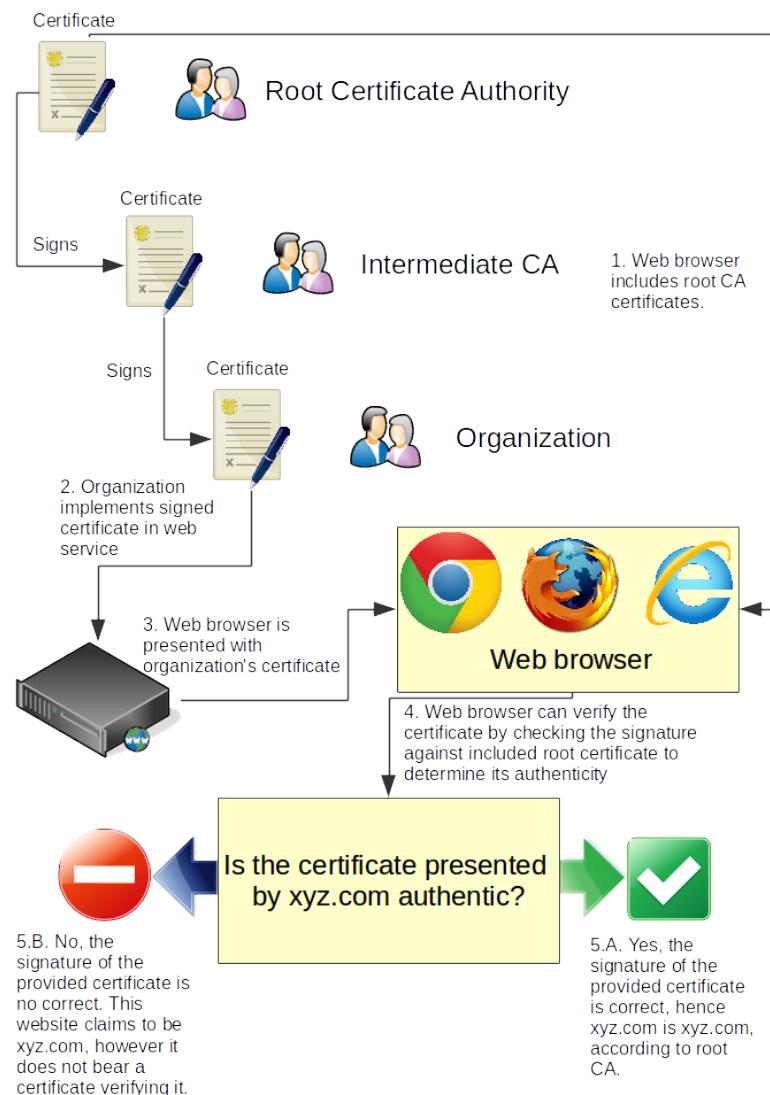


Figure 2.2: Public Key Infrastructure for certificates

When a user visits the Norwegian bank DNBs website (<https://dnb.no>), the website provides a certificate to the browser. This certificate claims that the website the user is currently visiting is <http://dnb.no>, and it is the bank DNB A/S who owns it. This certificate is digitally signed by a certificate authority which the browser trusts, and the browser will give the user some visual clue informing that the website being visited is the website it claims to be. Figure 2.3 illustrates how the URL bar in Firefox looks when the website of DNB is visited. As seen in the image, a green padlock combined with green text indicate two things. First, the connection towards the server is a secure, encrypted connection. Second, this secure connection has been established using a valid, verified certificate, which is issued for www.dnb.no.

I have previously in this chapter described what information a SSL/TLS certificate contains. In listing 2.2 an excerpt of the certificate presented to

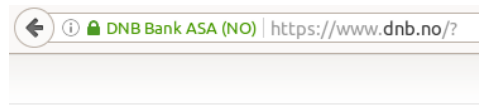


Figure 2.3: Firefox address bar indicating valid certificate

the browser when visiting www.dnb.no can be seen.

Listing 2.2: Digital certificate presented by <https://www.dnb.no>

```
marius@guile:/tmp/certs$ echo | \
openssl s_client -showcerts -servername www.dnb.no -connect www.dnb.no:443
2>/dev/null | \
openssl x509 -inform pem -noout -text
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      60:26:cd:70:58:a9:0f:1b:88:1c:d1:00:c7:40:f0:28
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C=US, O=Symantec Corporation, OU=Symantec Trust Network,
      CN=Symantec Class 3 EV SSL CA - G3
    Validity
      Not Before: Jun 22 00:00:00 2015 GMT
      Not After : Jun 21 23:59:59 2017 GMT
    Subject: 1.3.6.1.4.1.311.60.2.1.3=NO/businessCategory=Private
      Organization/serialNumber=984 851 006, C=NO, ST=Oslo, L=Oslo,
      O=DNB Bank ASA, OU=ITDIT, CN=www.dnb.no
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (2048 bit)
      Modulus:
        00:d5:3c:67:5e:35:57:14:89:84:cf:ad:f7:68:ca:
        ...
        4f:fd
      Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Subject Alternative Name:
        DNS:www.dnb.no, DNS:dnb.no
      X509v3 Basic Constraints:
        CA:FALSE
      X509v3 Key Usage: critical
        Digital Signature, Key Encipherment
      X509v3 Certificate Policies:
        Policy: 2.16.840.1.113733.1.7.23.6
        CPS: https://d.symcb.com/cps
        User Notice:
          Explicit Text: https://d.symcb.com/rpa

    X509v3 CRL Distribution Points:

      Full Name:
        URI:http://sr.symcb.com/sr.crl

    X509v3 Extended Key Usage:
      TLS Web Server Authentication, TLS Web Client
      Authentication
    X509v3 Authority Key Identifier:
      keyid:01:59:AB:E7:DD:3A:0B:59:A6:64:63:D6:CF:20:07:57:D5
      :91:E7:6A
```

```

Authority Information Access:
  OCSP - URI: http://sr.symcd.com
  CA Issuers - URI: http://sr.symcb.com/sr.crt

1.3.6.1.4.1.11129.2.4.2:
  ...i.g.v.....X.....gp
  ....N..t.....G0E..!..]/C.C'....2(o.t.pa.....C2.<..\_4.=./6L
  ....-...6..q...."K.@...v.V.../.....D.>.Fv....\....U.....N..t
  ....G0E..H..K....E.C..?.Y..>...W...U[.x....!...B...UP.o.=n..n.#.+...
  cEu...gr...u.h....d...:(.L.qQ]g..D.
v.Q.O.....N..t8.....F0D..H.h{....._...,*..I..G...'.Q.2...M.F.#HN..Fbp
<...7..r...O.f.Z
Signature Algorithm: sha256WithRSAEncryption
52:63:ea:27:98:7f:32:c4:1b:5d:99:88:3b:81:6a:cc:97:48:
...
4f:a1:29:30

```

Just as when the browser will tell users when a valid certificate is used, the web browser will warn users if they visit a website with a certificate which identity cannot be validated by the browser. Or, if the certificate is issued for some other domain than the user is visiting, as seen in figure 2.4. In this case a HTTPS enabled web server (nas.kaffemarius.no) was visited using Google's web browser Chrome, but the server is using a self signed certificate. This certificate has therefore not been signed by any of the root certificate authorities Chrome is trusting by default. In consequence the user is warned that the connection towards the server is encrypted and secure, but the certificate used to establish this connection is not trusted.

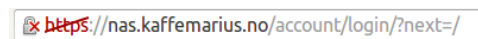


Figure 2.4: Chrome warning user that the certificate used is not signed by a trusted CA

For the SSL certificates to have any value, it is essential that the user actually verifies the domain name of the website he or she is visiting. For example, assume that a given user is using the Norwegian bank "Bank Norwegian". Bank Norwegian has an online banking system, and its web address is <http://banknorwegian.com>. A possible attack scenario would arise if an attacker crafted a website visually identical to the Bank Norwegian's. However, behind the scenes it is designed to steal user credentials. The attacker would register a domain name banknorenwegian.com, and point that domain name to the maliciously crafted website. In the fraudulent domain the position of the "w" and the "e" is swapped. The differences in the two domains are subtle and easy to overlook, but crucial. The attacker would also be able to get issued a valid, signed SSL certificate for the domain. This would cause the browser to provide the same visual cues to the user informing her that the visited website is in fact the website it claims to be.

Then the attacker could send an e-mail to the user informing him or her that they have received a transfer or that they will have to update their user account details, and encourage them to log in to the online bank. By embedding HTML in the e-mail, the attacker could include a link which read <http://banknorwegian.com>, while it actually was pointing towards banknorenwegian.com, making it even harder to detect. This could easily be

achieved by using the following HTML code:

```
<a href="http://banknorewegian.com">http://banknorewegian.com</a>
```

In an e-mail client which supports HTML rendering, the HTML snippet above would be displayed as seen in figure 2.5.

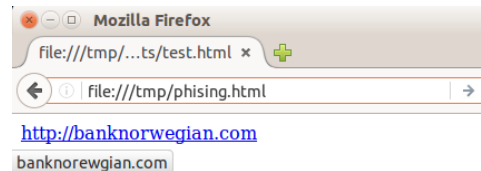


Figure 2.5: HTML phishing technique

Now the attacker has successfully lured users into visiting the maliciously crafted website. The users are now prompted for their username and password, which is sent to the attacker. Now the attacker has everything he or she needs to impersonate the user, and log in to the real Norwegian Bank online system, impersonating them. Attacks similar to the fictive example described above are common. In December 2013 the Norwegian bank “Sparbank 1” issued a warning towards its customers, raising awareness regarding current phishing campaign targeting their customers[4], which used the exact techniques which are described above. As we can see, the poor usability of the global SSL PKI mode makes it hard for users to identify and verify the websites they visit. Attackers exploit this by creating fraudulent phishing websites, and by sending phishing e-mails, they lure users into disclosing sensitive information. This could enable them to impersonate their identity online.

2.1.3 Data authentication

Sending information or data over the Internet is comparable to sending a regular letter using postal services. When sending a letter, one would typically write the message, and put it in an envelope before specifying the recipient. The first step towards getting the letter to the recipient, would typically be to put the letter into a mailbox where the postal service would transfer it further towards its destination. While being in the postal service’s possession, the letter would travel through several intermediaries such as postal terminals and similar before reaching its destination. It is important to point out that when it drops into the first mailbox, the original sender does not have any control over what happens to the letter. This is an important point from a security point. The sender does not have any guarantee that the letter will not be modified as being transferred on its way to the recipient.

The same rules apply when sending information over the internet. As soon as information leaves your computer, you have no control over what happens to it while being routed through the Internet. In many (or probably most) cases, one would like to have some sort of assurance that the information received by the recipient is correct and identical to what

was originally sent. In other words, we would like means to validate that the information is authentic. The simplest way to achieve this is to calculate a checksum of the message, and send that checksum together with the message. This would allow the recipient, when receiving the message, to calculate a checksum of the message received. The recipient could proceed to compare the two checksums, the one received with the message and the one calculated after the message was received. If the checksums are equal, the user would know that the message that was received, is identical to the message that was originally sent. If the checksums differ, the recipient would know that some kind of modification of the message has occurred since the first checksum was calculated, and that the message that was received is not authentic.

A common way to achieve this kind of authenticity, is by using hash functions. Please see Section 3.1.1 on page 35 for more details on hash functions. This approach to data authentication is efficient against unintentional modification. For example, if a postcard was accidentally ripped in two pieces while being transported by the postal service, and the recipient only received one of the pieces together with the checksum. In this scenario, the recipient could, by recalculating the checksum, learn that the message received was not authentic. However, if a dishonest employee in the postal service deliberately took the postcard, and modified its content, the dishonest employee could proceed to recalculate the checksum and replace it with the original all together, before sending the postcard onwards. In this scenario, the recipient would receive the message and the checksum, and by recalculating the checksum the message would appear authentic. In a situation where someone deliberately modifies the content of the message, the use of a regular hash checksum to prove authenticity is insufficient. Instead, this approach can be said to provide *integrity*, since it does provide a means for proving that the message is intact, and has not been modified by mistake.

In a situation where deliberate modification can occur, we need to improve our hash based scheme in order to detect whether a message is authentic or not. The simplest way to do that, is to introduce a secret key shared between the sender and the recipient. Instead of just calculating a hash of the message, we calculate a hash of a key, followed by the message itself. This can be expressed as $x = h(m||k_1)$. This is also known as a MAC, (Message Authentication Code), or *keyed hash*. When the message is received, the recipient, who also knows the secret key, can recalculate the hash using the key, followed by the message. If the message is authentic, the two hashes would match. If there is a mismatch between the hash calculated by the recipient and the hash received along with the message, the recipient would know that something was wrong. Since we assume that only the sender and the recipient know the secret key, no third party would be able to alter the message and recalculate the hash unnoticed. The only way to forge a message would be to identify a collision, which we assume is (when using a cryptographically secure hash algorithm) computationally infeasible.

By using message authentication codes, recipient *A* can prove to itself

that a message M sent from sender B is authentic or not. This can be said to prove *authenticity*. However, since both the sender and the recipient know what the secret key is, neither of them can prove to a neutral party, C , whether a given message originated from A or B . If we go back to our postcard example, we can imagine that the recipient receives a postcard containing a very insulting, hateful message together with the corresponding MAC. In this scenario it would seem likely that the sender simply sent a very nasty postcard, since the MAC received together with the message proves that the message is authentic. However, it could also be the case that the recipient sent the postcard to itself for whatever reason. Maybe the recipient wants sympathy, or even cause some legal troubles for the person who seemingly sent the postcard. The issue with message authentication codes in this case, is that neither the recipient nor the alleged sender is able to prove, to say the law enforcement, who actually sent the postcard.

In some cases it would be useful to be able to prove to a neutral third party where a given message originates from. This is what we know as "*non-repudiation*". In addition to our postcard example we can imagine other scenarios where non-repudiation is critical, such as when a customer registers a purchase while shopping online. In this way the customer cannot deny purchasing the goods. On the other hand, we would also like that the store is unable to deny receiving the order and payment, and thus protecting the customer. As we already have established, we are not able to achieve non-repudiation using the approaches previously discussed. In order to achieve this, we will have to utilize more complex approaches based on asymmetric cryptography.

Asymmetric cryptography utilizes two different keys, in contrast to symmetric cryptography where only one key is used. These keys are mathematically bound to each other, and their relation is based on mathematical problems which we assume are hard to solve, such as factoring large integers. Each entity in a asymmetric cryptosystem is assigned a pair of keys. One key, often called the private key, is to be kept secret from everyone except the owner. The other key, the public key, is shared with everyone.

Digital signatures are a class of algorithms based on asymmetric cryptography (please refer to Section 3.1.3) used to prove data origin authenticity. Depending on the mathematical problem each algorithm is based on, they operate differently. Common to all however is that one key, the private key, is used to generate the signature and the other key, the public key, is used to verify the signature. One popular digital signature scheme is the RSA digital signature, combining the RSA algorithm which is described in depth in section 3.1.4, and hash functions.

The RSA algorithm (please refer to section 3.1.4 on page 40 for more information) utilizes two keys for its operations: one key for encryption (k_1), and one key for decryption (k_2). These keys actually consist of two values each, the key itself and a special value used for the mathematical operations, called the modulus. The key is different for k_1 and k_2 , while the modulus is the same in both. When encrypting something with one of the

keys, the other key will have to be used for decryption. We keep one key for ourselves, k_1 . Then the other key, k_2 , is distributed so that the whole world knows about it. If we now revisit our postcard example, one might think that we could use this to prove authenticity just by encrypting the whole postcard using our secret key (k_1) before sending it. Then our recipient, or any other third party for that matter, could verify that the postcard originated from us, simply because the other key in the key pair (k_2) can be used to decrypt it. However, there is a technical limitation within the RSA algorithm itself which forces us to take an alternative approach.

The RSA algorithm cannot directly be used to encrypt/decrypt messages longer than the size of the modulus used. Currently the RSA Laboratories recommend a modulus size of 2048 bits⁷, which would then limit the message length to $< \frac{2048}{8} = 256$ characters, assuming one character is 8 bits in size, such as in the American Standard Code for Information Interchange (ASCII) character encoding set. One can assume that the length of postcard does indeed vary, but I am guessing that the vast majority of postcards will exceed 256 letters in length, which makes this fixed length limitation quite inconvenient. Of course, one could increase the size of the modulus (which in practice would be equivalent to increasing the key size), but that would also affect the performance of the encryption and decryption operations.

Luckily we already have a means to generate a fixed length signature of a message of arbitrary length: the hash functions. Instead of encrypting the postcard in whole before sending it, we could produce an SHA-1 hash sum of the message (which always will produce a 160 bit output, way below the modulus size of 2048 bits), and encrypt this hash sum using our secret RSA key. Then we could send the postcard together with the encrypted hash sum to our chosen recipient. Upon receiving our postcard and attached encrypted hash sum, the recipient would start by decrypting the hash sum. Then the recipient could generate an SHA-1 hash sum of the postcard, and compare that against the, now decrypted, hash sum received with the postcard. If the hash sums are equal, this tells the recipient (or any neutral third party for that matter) that the message is authentic, and that it must originate from us, assuming that the secret key k_1 has been kept safe. In addition, we could not deny sending the message, since the authentic hash was encrypted using our secret, private key, once again assuming that the secret key, k_1 , has been kept safe. In other words, we have now achieved what is known as non-repudiation.

2.2 Online banking and data authentication

Authentication, as described in section 2.1.3, refers to both identification (user and system authentication) and data-origin authentication (or simply data authentication). As this thesis focuses on data authentication of online

⁷How large a key should be used in the RSA cryptosystem - <http://www.emc.com/emc-plus/rsalabs/standards-initiatives/how-large-a-key-should-be-used.htm>

banking transaction data, we will now explore some of the technologies that have been, or are still being actively used today to solve this specific problem. BankID has been given special attention as it is the predominant technology used within the Norwegian banking industry as of today.

2.2.1 Transaction Authentication Number

Transaction Authentication Numbers (TAN's) are probably the most primitive way to attempt to ensure data authentication for online banking transactions. The bank will produce a list of valid TAN's, one for each customer. This list, typically physically manifested as a single sheet of paper containing numerous multi digit values, is distributed using mail or other means of transportation.

Whenever a customer registers a transaction within the online bank, the user is required to provide one of the TAN's found in the list issued by the bank. If the customer is able to provide a valid TAN together with the transaction details, the bank will proceed to process the requested transaction. Each TAN is usually only valid once, in order to mitigate replay attacks.

The use of TAN's does not really provide any data authentication at all. TAN's only prove that *someone* is in the possession of the list of valid TAN's when the transaction is registered. This could be the user, but it could just as easily be an attacker who has got hold of the list of TAN's. Further, since there is no connection between a given transaction and a TAN, a MitM attack taking place within the browser could just alter the transaction details behind the scenes, while attaching the valid TAN happily provided by the user before the request is made towards the bank. This is also known as a *Man-in-the-browser attack* (or MitB).

2.2.2 Indexed Transaction Authentication Codes

Indexed Transaction Authentication Codes (henceforth referred to as *iTAN's*) is quite similar to regular TAN's. Instead of just having a list of valid TAN's, each iTAN's is numbered, in other words, indexed. When a user attempts to register a transaction, he or she is required to provide a *specific* TAN from the list. iTAN's are, just like TAN's, susceptible to MitB attacks, and provide no additional security. The use of iTAN's does not provide any real way of proving the authenticity of a banking transaction.

2.2.3 Indexed TAN with CAPTCHA

As a further improvement of the TAN model, TAN's in combination with CAPTCHAs, also known as *iTANplus*, was developed. CAPTCHA is an abbreviation for "*Completely Automated Public Turing Test to Tell Computers and Humans Apart*", which are tests that can be generated automatically, which are simple to solve for humans, but hard to solve for computers [2]. An example of a CAPTCHA can be seen in figure 2.6. In this CAPTCHA, the text "smwm" is displayed, however both the image itself as well as



Figure 2.6: CAPTCHA displaying the text: “smwm”

the letters are distorted in a way which makes it difficult for a computer to interpret and extract the correct sequence of letters automatically using Optical Character Recognition (please refer to section 3.2 on page 42). For humans on the other hand, in most cases, identifying the correct sequence of letters is quite easy. This makes this test suitable for distinguishing humans from computers. Standard CAPTCHAs, as the one illustrated in figure 2.6, can be helpful when trying to limit automatic login attempts or similar, but does not provide any means to verify the authenticity of a given piece of data. In order to provide this, CAPTCHAs used in e-banking are generated based on the transaction details as well as a shared “secret” between the bank and the customer.

When a user registers a transaction, the server will automatically generate a CAPTCHA typically containing the transaction details, a piece of information which is shared between the bank and the customer, e.g., the customer’s birthday and a dynamic TAN. This image is then sent back to the user’s browser where it is displayed. Then the user can, by studying the image provided by the server, either approve or decline the transaction. The bank will only process the transaction when it has received approval from the user [18].

The iTANplus approach provides a means of verifying the authenticity of e-banking transactions. However, according to literature [18], methods exist for malicious software to alter the CAPTCHAs used for e-banking on the fly, which makes the technology susceptible to MitB attacks.

2.2.4 Mobile TAN

Mobile TANs (or *mTAN*’s) are instead of being printed on a sheet of paper, sent to the customer’s phone each time a transaction is requested. Whenever a customer registers a transaction, an SMS containing a TAN, together with some of the transaction details, such as the amount and recipient, is sent from the bank to the customer’s phone. In this way, the information is sent over an out-of-bands channel (the telecommunications networks), and is not directly susceptible to MitB attacks. A *mTAN* scenario is illustrated in figure 2.7, together with table 2.1.

Despite this, other ways exist to exploit this model, as was proven elegantly by the Eurograbber malware. A Eurograbber attack began by infecting an e-banking user’s computer with malware. This malware would then launch a phishing attack targeting customers of certain online banking systems. Attackers were then able to trick users to tell them their phone number and the phone platform they were using. Once the attackers were in possession of a customer’s phone number, an SMS message was

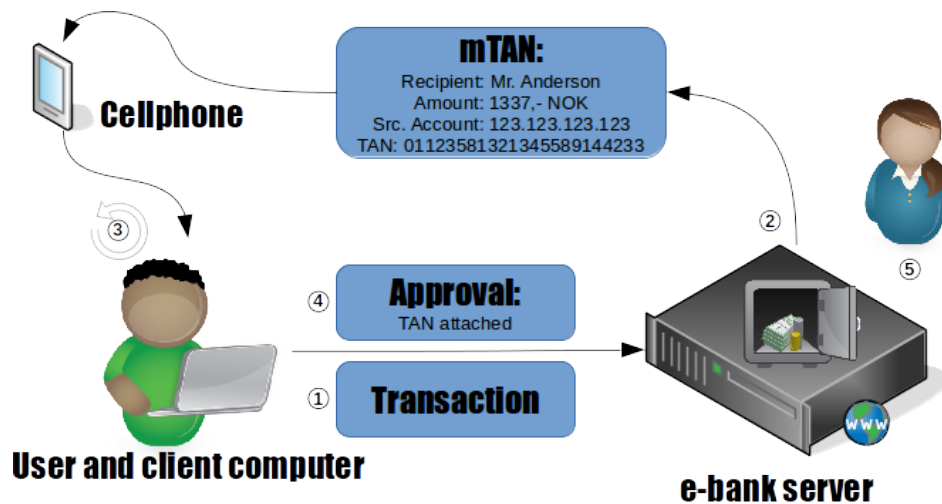


Figure 2.7: mTAN ceremony

Nr.	Message/action description
1.	User registers transaction in e-banking web application
2.	e-banking application sends mTAN message to users phone
3.	User verifies the details listed in the mTAN message
4.	User sends TAN as approval to e-banking application
5.	Transaction is processed when approval is received

Table 2.1: Sequence of messages and actions in the mTAN scenario

sent to their phone urging them to do a “security upgrade”, by installing a piece of software, for which a URL was included in the message. This software was no security update, but rather a nasty piece of malware capable of intercepting and altering incoming SMS messages before they become visible for the user. In this way, the Eurograbber malware was able to intercept the SMS containing the mTAN message, and alter it before notifying the user. As a consequence, users would approve arbitrary transactions made by the malware residing in his or her computer, instead of legitimate transactions registered by the user herself. As a result, an estimated value of more than 36 million Euro was stolen from over 30 000 users of different European banks [14].

2.2.5 TAN Generators

Instead of distributing TAN’s using a sheet of paper, some e-banking systems offer their customers TAN generators. These are small hardware devices which generate seemingly random numbers, either based on time synchronization or mathematical algorithms. TAN generators are initialized in a way so that it holds a secret shared only with the service provider it is configured to be used towards. This is essentially an OTP model, as described in section 2.1.1, except it is used for approval of e-banking transactions. That is, the OTP values are used as TAN’s. An RSA

SecurID OTP generator can be seen in figure 2.8.



Figure 2.8: RSA SecurID OTP generator

The idea is that when a user registers a transaction within the e-banking web application, the user will have to provide a valid TAN generated using the TAN generator in order for the transaction to be processed. Using TAN generators instead of regular TAN's (printed on sheets of paper) may improve the user experience, since they are simple to bring along due to their small size. However, in terms of proving the authenticity of banking transactions, they offer no more security than regular TAN's, and are susceptible to MitB attacks in the same way as regular TAN's.

2.2.6 photoTAN

photoTAN is an approach where the details of a given transaction alongside with a TAN are transferred to the customer's phone by using visual, computer readable data. When a customer registers a transaction, the e-banking web application generates a colourized Quick Response (QR) code. This QR code contains the transaction details as well as a generated TAN. The QR code is then scanned using a photoTAN smartphone application installed in the user's phone. The photoTAN application will decode and display the information found in the QR code in the smart phone's display. The user can then verify the transaction details on this display, and if the transaction is valid (that is, the transaction the user has requested), the user inputs the provided TAN in the web browser, and the transaction is processed [1]. This is illustrated in figure 2.9

The photoTAN approach allows the customers to use a secondary device when verifying the transactions details, and assuming that the smart phone can be trusted, provides a means of verifying the authenticity of e-banking transactions. However, as seen in section 2.2.4, the Eurograbber trojan has proved that this is not always the case. The photoTAN approach is therefore susceptible to advanced attacks where both the phone and the computer is compromised.

⁸Retreived from https://www.commerzbank.de/portal/media/a-20-themen/service-und-hilfe/hilfe-2/grafiken-99/App_2_scanmodus.jpg



Figure 2.9: Commerzbank photoTAN⁸

2.2.7 chipTan

The *chipTAN* approach also utilizes a separate hardware device allowing users to verify transactions before they are processed by the bank. A chipTAN generator is a small hardware device, featuring a small keypad, a display, a credit card slot and a camera, as seen in figure 2.10.



Figure 2.10: chipTAN generator⁹

⁹Retreived from <http://www-ti.informatik.uni-tuebingen.de/~borchert/Troja/Bilder/chipTANcomfort.jpg>

As in the previous approaches, the process begins when a customer registers a banking transaction within his or her e-banking provider. The e-banking system receives the transaction, and generates a specially crafted animation which is to be interpreted by the chipTAN device using its camera. The chipTAN device is locked to a user's credit card, and will not operate unless the specific credit card is present in the chipTAN's credit card slot. When the specific credit card is present, the user is allowed to operate the device to scan the animation shown on the computer's display. The transaction details are embedded in the animation, which when interpreted using the built-in camera, is displayed using the chipTAN's display. The user can verify that the transaction is valid using the chipTAN, and request that a TAN is generated. This TAN, now being displayed using the chipTAN, is sent to the bank if the user wishes to approve the transaction [26].

The chipTAN approach provides a simple and effective way to verify e-banking transactions. It is not susceptible to MitB attacks, since the transaction is verified using a separate hardware device. In contrast to the photoTAN approach (section 2.2.6), the chipTAN model is much harder to compromise, since the attack surface is limited to the built-in camera (at least for non-physical attacks). This makes it resilient against attacks similar to the previously described Eurograbber attack and similar attacks.

2.2.8 BankID

BankID is a service provided by a coalition of Norwegian banks, using a shared infrastructure maintained by Nets. BankID offers a way for users to identify themselves, as well as sign electronic documents online. BankID is implemented as a PKI (Public Key Infrastructure), and every registered user is assigned a personal certificate stored centralized within the system itself. In addition to being used for verification of e-banking transactions, the BankID system can be used for user authentication towards a broad range of public services such as health services, pension and social services and tax services through a common login platform *ID-porten* provided by Difi (Agency for Public Management and eGovernment) [25]. As seen in figure 2.11, skatteetaten.no, the portal for tax services supports user authentication both by using BankID and BankID on mobile.

The BankID service is the predominant technology used to approve online banking transactions in Norway, and is used in one way or the other by 3,5 million Norwegians [3]. Despite its broad adoption, there is little literature available describing how it works in detail, which is in the nature of a proprietary system such as BankID. In this way, it is not possible to provide an accurate description of how the system operates. However, based on details which are publicly known, I will try to provide a general outline of how it *could* be operating. It is important to note that the descriptions found in the following sections are based on my own experience with BankID used in combination with one e-banking system in particular (<https://dnb.no>). Hence, the procedures could differ when BankID is used in combination with other e-banking systems.

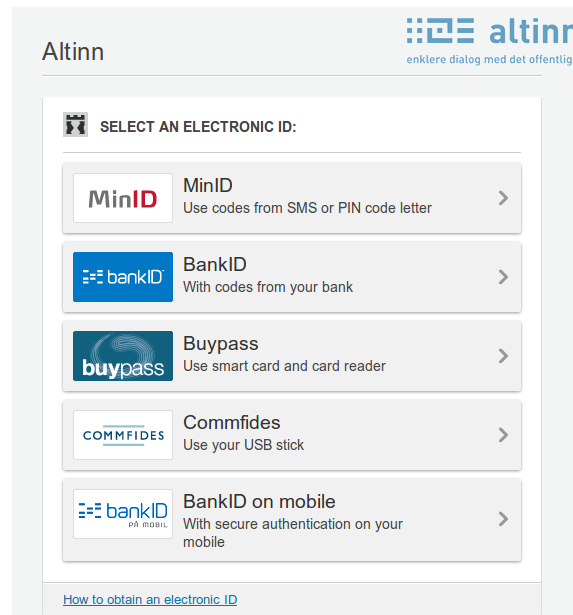


Figure 2.11: Login alternatives for skatteetaten.no

BankID offers two approaches for approving e-banking transactions. One is based on a traditional approach using an OTP generator and a password. In addition, BankID offers an implementation for smartphone users, which involves an application running from the SIM card for improved security.

BankID

The traditional *BankID* approach requires a password in addition to a one-time password generated using a OTP/TAN generator or smart card when being used for either identification, authentication or signing. Thus, this approach is similar to the other TAN-/OTP-based approaches previously discussed. When a user has requested a transaction in an online banking system offering BankID, the user will have to confirm or verify the transaction by entering his or her password and a generated one-time password before the transaction is processed.

Since the BankID is a standalone system, hosted and operated independently from each individual e-banking system, it seems likely that when a user verifies a transaction in the browser, the browser will communicate both to the e-banking system, as well as directly to the BankID system. Since BankID handles the certificated used to authenticate/sign the transaction, there will have to be some sort of communication between the BankID system and the e-banking system transparent from the user's perspective. This work flow is illustrated in figure 2.12, together with table 2.2.

From a theoretical perspective, this approach is susceptible to a MitB attack, in the same way the TAN based approaches previously discussed.

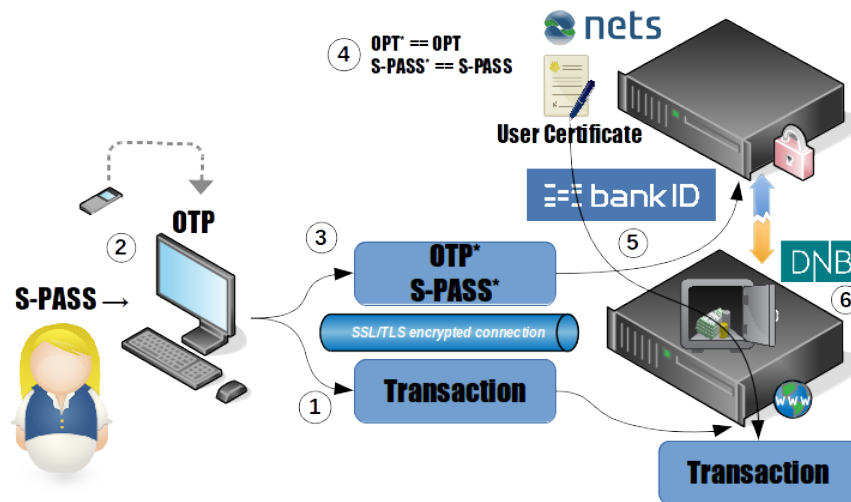


Figure 2.12: Illustration of how BankID *could* be implemented

Nr.	Message description
1.	User registers transactions
2.	User provides OTP from OTP generator and static password
3.	OTP and static password is sent towards BankID, together with transaction for authentication
4.	BankID verifies that the OTP and static password are correct
5.	If OTP and static password is correct, the transaction is signed using the user's certificate stored within BankID. Signature is sent to bank
6.	When bank receives valid signature from BankID, the transaction is processed

Table 2.2: Possible BankID work flow

BankID on mobile

BankID on mobile operates in a different manner. The approach requires the user to have a SIM card preloaded with the a BankID application. This tells us that the BankID on mobile is implemented as an SIM Application Toolkit (often referred to as SAT) application. These SAT applications run directly from the SIM card itself, using the phone's SIM CPU, which is separate from the CPU (or CPUs) used to run the operating system on the phone itself.

When a user has acquired an BankID enabled SIM card, it must be initialized. This is usually done within the e-banking web interface when the user is already logged in. In order to initialize the SIM card for a specific user, the user provides his or her phone number. When the user's phone number has been provided, a message is sent (presumably from the central BankID system) towards the phone, which initiates the setup process. The initialization process is both quick and simple, the user is only asked to choose a static PIN, four to eight digits in length on the handset before it is

ready for use.

Once BankID on mobile is initialized, it can be used both for user authentication, as well as transaction authentication. When a user chooses to use BankID on mobile for user authentication, the user's phone number first has to be provided in the e-banking system web interface, as seen in figure 2.13. When the phone number has been provided, an image

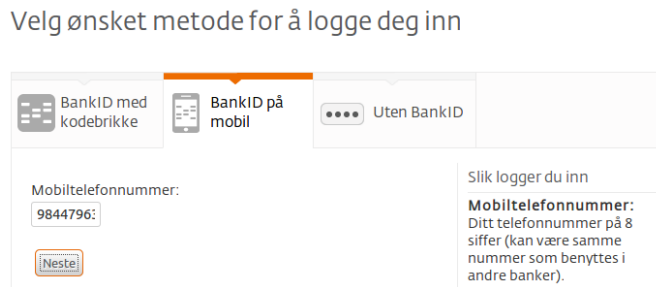


Figure 2.13: BankID on mobile splash screen

containing two code words is being displayed in the browser window, as seen in figure 2.14. The same two code words are then displayed on the

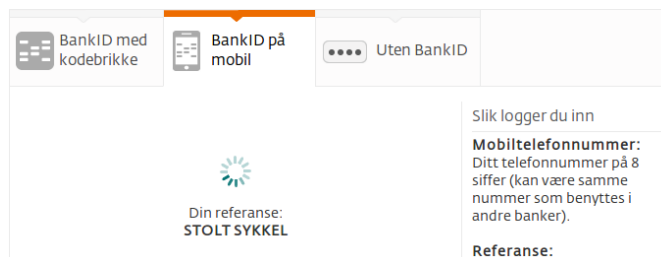


Figure 2.14: Code words for BankID on mobile being displayed in browser

user's handset using the BankID SAT application, as seen in left side of figure 2.18. If the user is able to verify that the same words are being displayed in both the browser window and on the handset, the user can now provide his or her static PIN to the SAT application, as seen in the right side of figure 2.18. This will cause the user to be logged in, and the web interface will update accordingly.

The procedure for authentication of transactions using BankID on mobile is quite similar. First, a user registers a transaction within the e-banking system web interface. When the transaction is registered, the user must press a "Confirm transaction" button within the web interface. When pressed, the web interface will update itself and display the transaction details of the transaction in question, as seen in figure 2.16. The user is now asked to compare the details displayed in the computer screen to the details which are about to be displayed on the user's handset.

A message is now sent to the user's handset (presumably from the central BankID system) which will cause the BankID SAT application to be loaded, and displayed in the phone's display, as seen in figure 2.17. If

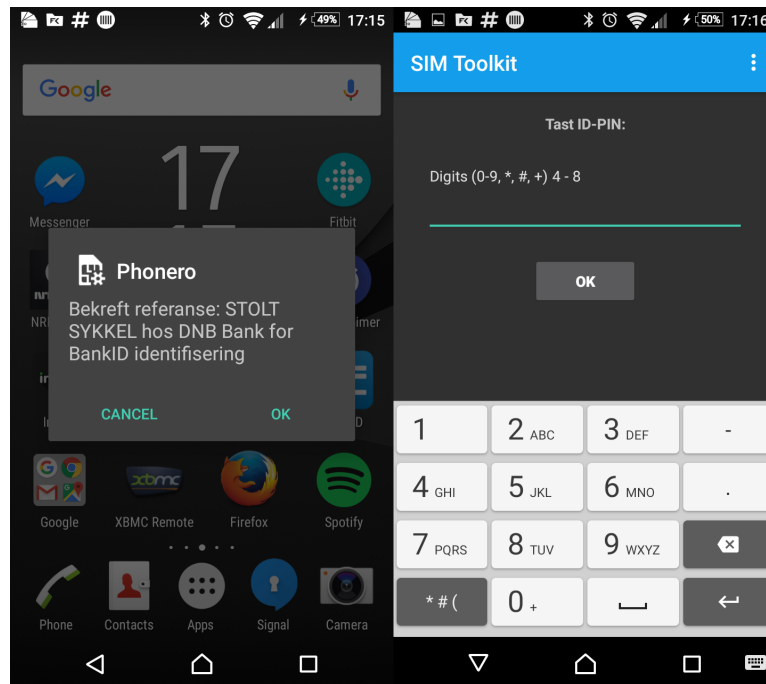


Figure 2.15: User authentication using BankID on mobile

the user presses “OK”, the transaction details are displayed in the display, as seen in the left side of figure 2.18. The user can now compare the details being displayed in the web browser to the ones being displayed in the phone’s display. If the user finds the information to be correct, he or she can authenticate the transaction by providing his or her static PIN, as seen in the right side of figure 2.18. When the correct PIN is provided, a message will be sent from the phone to what is presumably the central BankID system. The central BankID system will then notify the e-banking system that the transaction has been authenticated, and the web interface will automatically update accordingly. The transaction is now processed by the e-banking system.

BankID on mobile seems like a robust solution for data authentication in the e-banking use case. By utilizing a SAT application running directly on the user’s SIM card, users of this approach do not have to carry an extra hardware device around (assuming that most users already carry a phone anyway). Since the application runs separately from the operation system of the phone itself, it seems likely that it would be much harder to compromise this scheme compared to other schemes involving mobile phones, such as the Mobile TAN approach described in section 2.2.4 on page 24, as the attack surface is greatly reduced. This makes this system resilient against MitB attack scenarios.

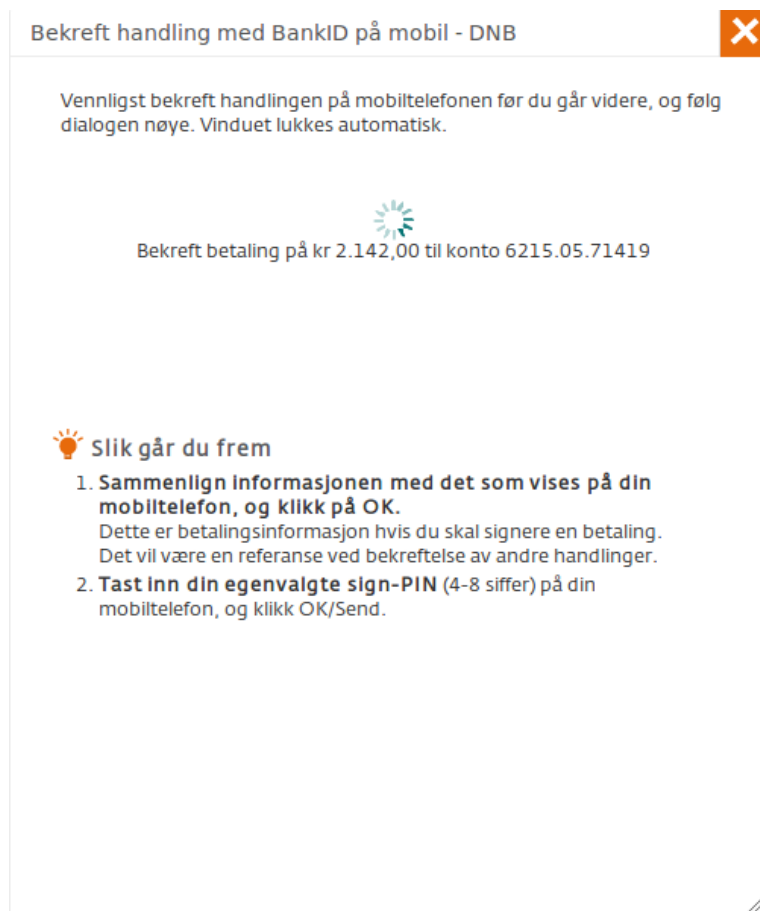


Figure 2.16: Transaction details in web interface

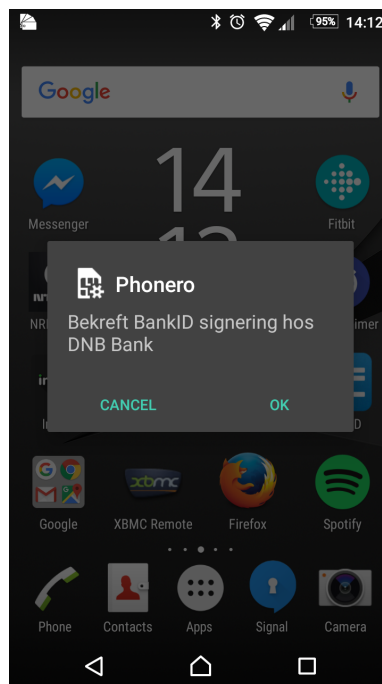


Figure 2.17: BankID SAT application – Splash screen

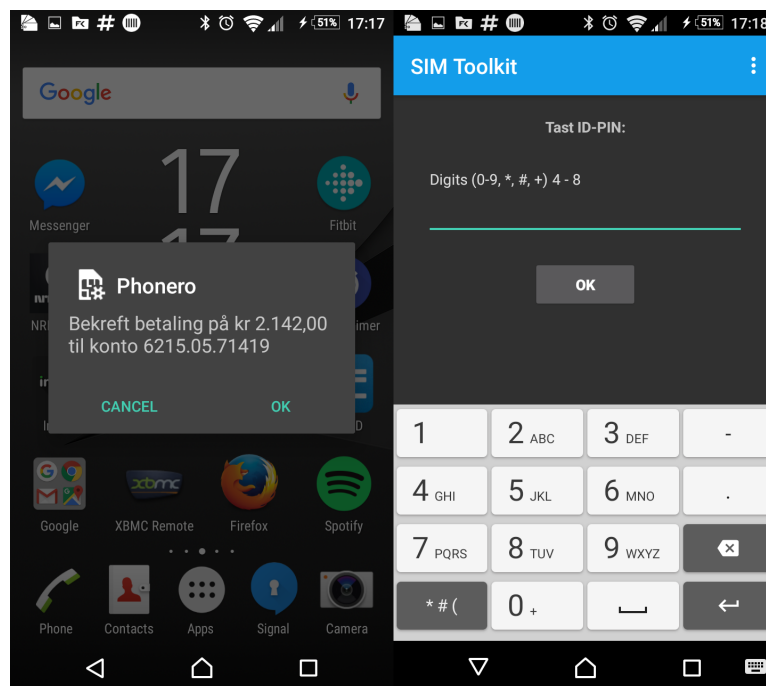


Figure 2.18: BankID SAT application – Authentication of transaction

Chapter 3

Technical background

3.1 Cryptography

This chapter contains more in-depth descriptions of some of the concepts and technologies referred to throughout this thesis, and can be used as a supplement while reading the subsequent chapters.

3.1.1 Hash functions

A cryptographic hash function, h , is a mathematical function which takes a message of arbitrary length, and produces a fixed length output deduced from the message, often referred to as a message digest. This could be expressed as $h(m) = x$, as illustrated in figure 3.1.

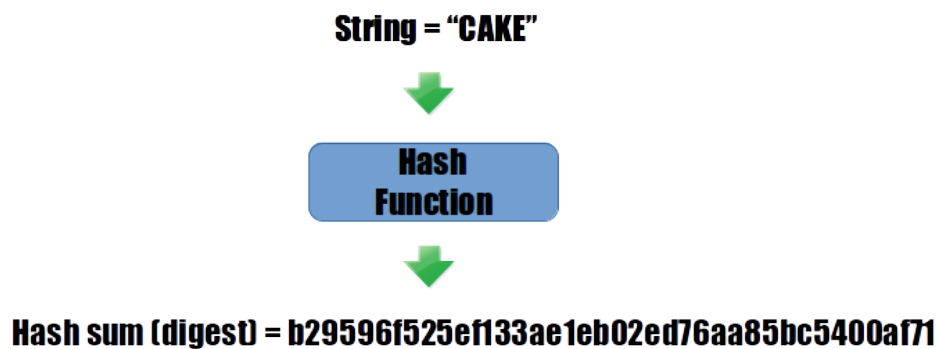


Figure 3.1: Hash function (SHA1) input and output

Trapp & Washington propose the following requirements [33, p. 218] which hash functions used in cryptography should satisfy:

1. For any message, m , the digest, $x = h(m)$, can be calculated quickly.
2. For a given x , it is computationally infeasible to find an m' where $h(m') = x$. In other words, the hash function should be a one-way function [20, p. 6] where it is computationally infeasible to deduce the message from the digest. When a hash function satisfies this

requirement, it is said to be **pre-image resistant**. This property is called **pre-image resistance**.

3. It should be computationally infeasible to find any two messages, $m_1 \neq m_2$ where $h(m_1) = h(m_2)$. If two different messages produce the same message digest, this would be what is called a collision. So if a hash function satisfies this requirement, it is said to be strongly collision-free. We call this property **collision resistance**.

Since hash functions output a finite number of symbols, there is an infinite number of messages which would produce the same message digest. Requirement three only states that it should be infeasible to find messages which produce the same message digest. In most cases it would be adequate if it is computationally infeasible to find a message, m_2 , for a *given* message, m_1 , where $m_1 \neq m_2$ and $h(m_1) = h(m_2)$. This is referred to as **second pre-image resistance** [33, p. 219], and when a hash function satisfies this requirement, it is said to be weakly collision free. At first glance this might seem as a repetition of the collision resistance property. In terms of collision resistance, the definition says that it should be infeasible to find *any* two messages, $m_1 \neq m_2$, where $h(m_1) = h(m_2)$. In terms of second pre-image resistance, we have one *specific* message, m_1 , and it should be infeasible to find another message, $m_2 \neq m_1$, where $h(m_1) = h(m_2)$.

In order to fulfil the three requirements described above, mathematical hash functions (or algorithms) are carefully constructed using primitive operations such as bitwise AND (\wedge), bitwise XOR (\oplus), modulo (mod), bitwise OR (\vee) and block shifting (\ll or \gg), where every block consists of n bits. The inner workings of most hash algorithms are quite complex, and is beyond the scope of this thesis. A range of different hash algorithms exists such as **MD5**¹, **SHA**² and **RIPEMD**³.

3.1.2 Symmetric cryptography

Symmetric cryptography is an umbrella term for cryptographic techniques where the same key is used for both encryption and decryption. One of the oldest known encryption schemes is called “*Caesar cipher*”, and is a simple example of a symmetric cryptosystem. In this system, every letter in the message (also known as the *plaintext*) is shifted n number of characters to the left according to its alphabet, in order to be encrypted. In order to decrypt an encrypted message (also known as the *ciphertext*), each letter in the ciphertext is shifted n number of characters to the right. If the English alphabet is encoded such that $A = 0, B = 1 \dots Z = 25$, we end up with the following table:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

¹MD5 - <https://tools.ietf.org/html/rfc1321>

²SHA - <https://tools.ietf.org/html/rfc3174>

³RIPEMD - <https://www.cosic.esat.kuleuven.be/publications/article-317.pdf>

This allows us to formulate the encryption process mathematically as $encryption(m, n) = m + n \bmod 26$, where m is the letter to be encrypted and n is the key. The decryption process can be described as $decryption(m, n) = m - n \bmod 26$. $\bmod 26$ is used in both calculations since $(25 + 1) \equiv 0 \bmod 26$. If one were to encrypt the message "APPLE" using the key 7, using "Caesar Cipher", one would end up with the following result:

`encrypt('APPLE', 7) = HWWSL`

Similarly, the decryption of the ciphertext would yield the following result:

`decrypt('HWWSL', 7) = APPLE`

From a security and cryptanalytic standpoint, The "Caesar cipher" is completely insecure, and is only used to illustrate what a symmetric cryptosystem looks like. Modern symmetric cryptosystems, such as AES⁴ and Blowfish⁵ utilize far more advanced mathematical principles which is a topic far beyond the scope of this thesis. The key point is that a single key is used for both encryption and decryption. This key has to be pre-distributed to both the sender and the recipient prior to any message encryption/decryption. In consequence, anyone with knowledge of the key is able to both encrypt and decrypt messages. For most situations, this is not necessarily a bad thing. However, if a situation requires that one is able to prove who encrypted a given message, a symmetric cryptosystem is unable to fulfil this requirement.

Considering the previous example where the message "APPLE" is being encrypted using "Caesar cipher" and the key used is 7. Assuming there are two parties, A and B , where A wants to encrypt the message "APPLE", then send it to B . Both A and B will have to have knowledge of the secret key, $k_1 = 7$. A will need it to encrypt the message, and B will need it to later decrypt the message. If a third party, such as C is introduced, there is no way for A and B to prove to C where the encrypted message originated from, since both have knowledge of k_1 . The ability to produce this proof, is what is called "*non-repudiation*" [19]. In order to achieve this kind of message origin authentication, another class of cryptography is needed; asymmetric cryptography.

3.1.3 Asymmetric cryptography

In contrast to symmetric cryptography where the same key is used for both encryption and decryption, asymmetric encryption (sometimes referred to as public key encryption) schemes utilize two different keys: one for the encryption process and another for the decryption process. This can be achieved by utilizing what is called "*trap-door functions*" in mathematics. As described in section 3.1.1 on page 35, one-way functions are mathematical functions where it is computationally infeasible to find its

⁴AES - <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>

⁵Blowfish - https://www.schneier.com/academic/archives/1994/09/description_of_a_new.html

inverse. Consider a one-way function f , it is easy to compute $x = f(y)$, but is hard to compute $y = f^{-1}(x)$. A trap-door function is similar to a one-way function, but it differs in the way that those who know where the secret lever is located, are able to escape the dungeon. For a trap-door function, tf , it is easy to compute $x = tf(y)$, but hard to compute $y = tf^{-1}(x)$, unless we have knowledge of some piece of secret information (the lever), k . If we have knowledge of k , it is easy to compute $y = tf^{-1}(x, k)$.

According to the fundamental theorem of arithmetic, every natural number (all numbers greater than 1, which is not a prime itself) is a product of a unique prime factorization [34]. For example, $8 = 2 \cdot 2 \cdot 2$. Be that as it may, the theorem does not provide any method as to how these factorizations are found. As of today, no efficient algorithms for finding the prime factors for very large integers using conventional computers are known. Therefore, this is assumed to be computationally hard and is known as the “integer factoring problem”. This can be used to illustrate a trap door function. Assume that only pen, paper and a simple calculator are available. Multiplying two primes, 113 and 367 using the calculator is a simple task, in terms of computational complexity. Within less than a second we learn that $113 \cdot 367 = 41471$. Reversing this process, that is, decomposing the composite number into its prime factors knowing only the final product is on the other hand believed to be very hard. The integer 1689025133569, is a product of two primes, and therefore a natural number. Finding out which two prime factors this integer consists of is a daunting task using just pen and paper and a calculator. As there is no simple way to deduce the prime factors from the final product, one would have to try multiplying combinations of all primes, $2, 3, \dots, \frac{1689025133569}{2}$. This is what is called a *brute-force* approach (sometimes known as an exhausting key search), as it in a worst-case scenario would imply testing of every possible combination (traversal of the entire key space). However, if it was revealed that one of the two prime factors is in fact 1299541, it would be easy to (by using a calculator) see that $\frac{1689025133569}{1299541} = 1299709$. This single piece of information makes it easy to find the second prime factor.

The example described above is only valid in a non-computer scenario, due to the fact that a computer would have had no problem to find the two prime factors in the given integer using a brute force approach. There even exist several algorithms to factor composite numbers which are more efficient than a brute-force approach. However, given a large enough integer, this becomes infeasible for even the fastest computers and most efficient algorithms publicly known to this date. In this context a large integer is typically greater than 1024 bits. The current record for integer factorization was obtained in 2009 when a group of researchers over the course of three years were able to factor a 768 bit (or 232 digits) integer. The computational resources used for this task is equivalent to 2000 years of computing on a common single core 2.2 GHz processor computer [15].

Although the previous example is a nice way to illustrate a trap-door function, it does fail to address how one can use these functions

as a foundation for asymmetric cryptosystems. In 1976 Withfield Diffie, Martin Hellman and Ralph Markele described a system for key agreement over insecure communication channels. The system is now known as the “*Diffie-Hellman key exchange*”⁶ (or D-H). D-H allows two or more entities to agree on a shared secret key despite the assumption that any communication between them is intercepted by a third party. This system is based on another mathematically problem called the “discrete logarithm problem”.

Given the equation $b^k = g$, where the values b and g are members of a finite group, G . The k which solves this equation is called the discrete logarithm. When big enough numbers are used, this is assumed to be very hard.

Assume Alice and Bob want to establish a shared secret key. They are both in the same room as Eve, so Eve can hear everything Alice and Bob say to each other. First Alice and Bob agree to use the base $b = 3$ and modulus $p = 31$. Alice and Bob then proceed to perform the operations described in figure 3.1.3. The figure also illustrates what information is visible for the intercepting entity, Eve. As seen in the figure, if Eve was to intercept the

Step	Alice	Eve	Bob
1	Alice chooses the secret key 9		Bob chooses the secret key 12
2	Alice computes $A = 3^9 \bmod 31 = 29$		Bob computes $B = 3^{12} \bmod 31 = 8$
3	Alice sends A to Bob	$A \rightarrow$	
4		$\leftarrow B$	Bob sends B to Alice
5	Alice computes $S = B^9 \bmod 31 = 4$		Bob computes $S = A^{12} \bmod 31 = 4$
6	Alice's $S = 4$		Bob's $S = 4$
7	Alice encrypts m using secret key S	$m^{enc} \rightarrow$	
8		$\leftarrow m^{enc}$	Bob encrypts m using secret key S

Table 3.1: Key agreement using Diffie-Hellman key exchange

initial key establishment denoted in step 1 to and including step 6, she only learns about A and B . For Eve to figure out the secret key shared between Alice and Bob, she will have to find the k which solves $3^k = 29 \bmod 31$ or $3^k = 8 \bmod 31$. As previously described, this is referred to as finding the discrete logarithm. In this example, we are using quite small numbers in order to illustrate how the protocol operates. Given the values used in this example, it would be feasible for Eve to find k using a brute-force approach with ease. However, when the numbers used are big enough, finding the discrete logarithm is assumed to be very difficult.

⁶In "An overview of public key cryptography" (2002) Martin Hellman suggested that the algorithm should be called "Diffie-Hellmann-Merkele key exchange, in order to emphasise Markele's contribution.

Now Alice and Bob have one shared secret key. This key can now be used in a symmetric cryptosystem, allowing Alice and Bob to secure communication between them over an insecure channel.

As with the previous example, the numbers used in this example are too small for any practical use.

Today D-H and closely related systems for asymmetric key exchange are widely used to establish the key used in HTTPS (HTTP over SSL/TLS), which allows web traffic to be encrypted between clients and servers.

As described in this section, mathematical problems exist which are hard to solve, even infeasible for the fastest of supercomputers. These problems can be used to construct asymmetric cryptosystems, such as the Diffie-Hellmann key exchange. In the next section, Section 3.1.4 on page 40, it will become evident how the integer factorization problem is the basis for one of the most recognized asymmetric cryptosystem of our time.

3.1.4 The RSA algorithm

The RSA algorithm was invented in the academic community by Ron Rivest, Adi Shamir, and Leonard Adleman, and published in a research paper in 1977. Thus, RSA is an acronym for Rivest, Shamir and Adleman. However, as revealed in 1997 when documents released by CESC (Communications-Electronics Security Group), a British governmental agency, showed that Clifford Cocks had described an algorithm very similar to the RSA algorithm as we know it today in an internal document written in 1973 [33, p. 164-165]. The RSA algorithm utilizes two different keys (one for encryption and a separate one for decryption), which makes it an asymmetric cryptosystem.

Euler's Totient Theorem lays the foundation for the RSA algorithm. This theorem states that given two integers, x and y , which are relatively prime, that is their GCD (Greatest Common Divisor) is 1. If x is less than y , then x in the power of Euler's totient⁷ of y is congruent to 1, modulo y . This theorem can be formalized in the following way: that if $\gcd(x, y) = 1$ and $x < y$, then $x^{\phi(y)} \equiv 1 \pmod{y}$.

According to the law of exponents, we can expand this rule. The expression $x^{\phi(y)} \cdot x^{\phi(y)} \equiv 1 \cdot 1 \pmod{y}$ can be rewritten as $x^{(\phi(y)+\phi(y))} \equiv 1 \pmod{y}$. This can be further simplified by rewriting it as $x^{(2 \cdot \phi(y))} \equiv 1 \pmod{y}$. Since $x^{(3 \cdot \phi(y))} \equiv 1 \pmod{y}$ we can deduce an even more general expression $x^{(i \cdot \phi(y))} \equiv 1 \pmod{y}$, where i can be any number.

In this way, we see that $i \cdot \phi(y)$ generates the set of integers, S , which are divisible by $\phi(y)$, $n \mid \phi(y)$. We can then rewrite the statement from the previous paragraph in the following manner: $x^{n \in S} \equiv 1 \pmod{y}$. Since $x^{n \in S} \cdot x \equiv 1 \cdot x \pmod{y}$, $x^{S+1} \equiv x \pmod{y}$, we see that we have a function which takes a number x , raises it to a power of $S + 1$, and when the calculation is done in \pmod{y} , we end up with the same number, x , which we started out with.

⁷Euler's totient function returns the number of positive integers between 1 and $< n$ which are coprime to n . This function is denoted as $\phi(n)$

If we find two values k_1 and k_2 , so that $k_1 \cdot k_2 \equiv 1 \pmod{\phi(y)}$. Then $x^{k_1 \cdot k_2} \equiv x \pmod{y}$, which also can be expressed as $(x^{k_1})^{k_2} \equiv x \pmod{y}$. The calculations in the last expression can be divided into the two separate steps:

Step 1: $x^{k_1} = c \pmod{y}$.

Step 2: $c^{k_2} = x \pmod{y}$.

If we imagine that the value x holds is some piece of data some entity would like to securely transmit to us, we can utilize **Step 1** as the encryption process, and **Step 2** as the decryption process.

This can be illustrated by using some sample values. First we chose two different prime numbers, $p_1 = 7$ and $p_2 = 11$. Then we calculate $y = p_1 \cdot p_2 = 77$. We use two prime numbers for two reasons. Firstly, we want y to be hard to factor. Secondly, we can quickly calculate $\phi(y)$ since p_1 and p_2 are primes, using the following formula $\phi(y) = (p_1 - 1) \cdot (p_2 - 1) = 60$. Now we need to find the appropriate values for k_1 and k_2 , so that $k_1 \cdot k_2 \equiv 1 \pmod{\phi(y)}$. We start with the value 7, so that $k_1 = 7$. Now we have to solve k_2 for the equation $7 \cdot k_2 \equiv 1 \pmod{\phi(y)}$. To make this process a bit more simple, we can, as previously described, rewrite the equation as $7 \cdot k_2 = i \cdot \phi(y) + 1$, or $7 \cdot k_2 = i \cdot 60 + 1$, where i can be any number. After some trial and error, we see that $7 \cdot 43 = 301$ which is equal to $5 \cdot 60 + 1 = 301$, so that $7 \cdot 43 = 5 \cdot 60 + 1$. Now we have our values for $k_1 = 7$ and $k_2 = 43$. These, together with $y = 77$ make up the keys in the RSA algorithm. k_1 and y represents the private key. k_1 is therefore kept secret. k_2 together with y constitutes the public key. In other words, k_2 is known by anyone.

Imagine a scenario where someone would like to send us a message only we would be able to read. Since both k_2 and y is available, anyone could create such a message. However, only those of us who have knowledge of k_1 are able to decrypt the message. Let us imagine that someone wants to encrypt the string "CAKE" and send it to us. As previously described, Euler's theorem states that x has to be $< y$, as well as $\gcd(x, y) = 1$. Due to this, we generate a list of values which fulfils these requirements and assigns one for each letter in the alphabet. This list can be seen in listing 3.1. Any value in the range $2 \cdot \dots \cdot (y - 1)$ which is not assigned to a letter, is denoted as "?".

Listing 3.1: Alphabet encoded using appropriate values

2	3	4	5	6	8	9	10	12	13	15	16	17	18	19	20	23	24	25	26	27	29	30	31	32	34	N
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	?

Using this list, each letter in the string "CAKE" corresponds to the values 4,2,15,6. These values can be encrypted separately using the the equation described in **Step 1**:

$$C = 4^{43} = 77371252455336267181195264 = 53 \pmod{77}$$

$$A = 2^{43} = 30 \pmod{77}$$

$$K = 15^{43} = 64 \pmod{77}$$

$$E = 6^{43} = 62 \pmod{77}$$

After these calculations, we end up with the values 53, 30, 64 and 62. According to the list we have generated, this would correspond to the

string “?W??”. This will now be referred to as the *ciphertext*, or encrypted data. Now, imagine these values were sent to us using an insecure channel, where some third party was able to acquire a copy of the ciphertext. The values, 53, 30, 64 and 62, translate to the string “?W??”, which does not make much sense to anyone. However, when we receive the message, we can utilize the previously described equation in **Step 2**, to recover the plain text:

$$? = 53^7 = 1174711139837 = 4 \pmod{77}$$

$$W = 30^7 = 2 \pmod{77}$$

$$? = 64^7 = 15 \pmod{77}$$

$$? = 62^7 = 6 \pmod{77}$$

Again, by using the list defined in listing 3.1, we build the string “CAKE”, which is the string that was originally encrypted by the sender.

The implementation of the RSA algorithm explained in this text is known as “*textbook RSA*” or “*schoolbook RSA*”, as it from a cryptanalytic perspective is completely insecure. Since each letter is encrypted one by one, the approach is susceptible to traditional cryptanalysis, or an attacker could just generate the whole alphabet in advance, since the public key is known by everyone. In this case, an attacker would have no problem decrypting the encrypted message. In order to mitigate this, professional grade implementations of the RSA algorithm implement padding schemes which removes this deterministic behaviour. Further, the values used in this example are way too small for the system to be secure. Just by using a pen and paper, it is simple to find the two prime factors for the 77, the value used for y . As previously described, the security of RSA is based on the assumption that it is computationally infeasible to factor large composite numbers. “*Large composite numbers*” in this context is order of > 2024 bits, which is RSA’s currently recommended key size for the RSA algorithm, as described previously in section 2.1.3 on page 19.

3.2 Optical Character Recognition

Optical Character Recognition (OCR) is the process of transforming visual text into digital information. OCR software is able to accomplish this by analysing the shapes of graphic letters in images of text, and transform them into alphanumeric characters in digital form [5], as seen in figure 3.2. In this figure, I have taken a picture of some made-up transaction details on a computer screen. This image is then transferred to a computer, where it is processed using an OCR engine. As seen in the figure, the OCR software is able to convert the text found in the image, into its digital representation with high accuracy. Automatic, computer-based OCR can be used for a wide variety of applications, including digitalization of documents [7][p. 177], license plate recognition [32] and text-to-speech aid tools for the vision impaired [10]. Optical Character Recognition technology has improved immensely since 1912 when Emmanuel Goldberg first invented a machine which was able to transform typed messages into telegraphic messages. Nearly forty years later, in 1951, David Shepard presented an OCR machine

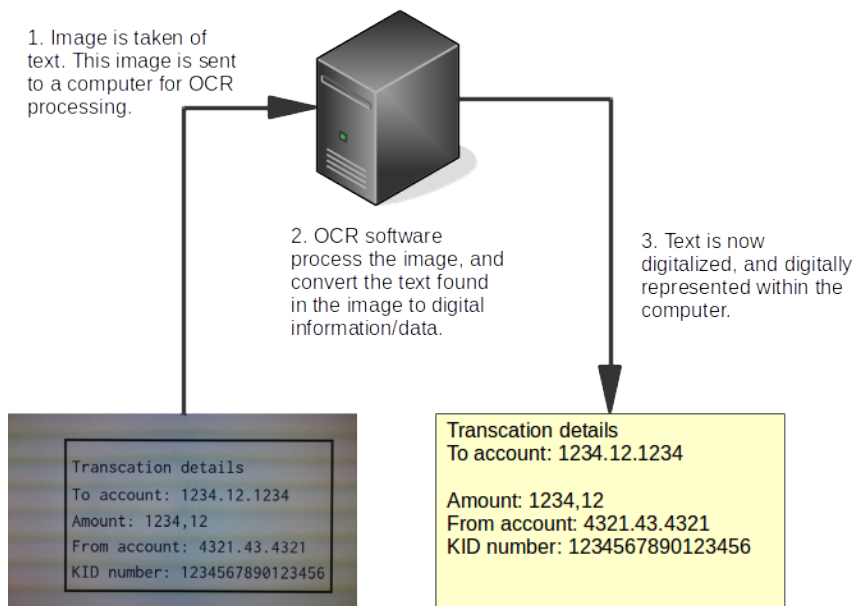


Figure 3.2: OCR overview

able to read just over twenty characters from a text written on a typewriting machine [7]. During the 1960s OCR was adopted to help solve everyday issues, such as sorting letters based on city/ZIP code within the postal industry [31]. Today, OCR technology comes in different shapes and forms, ranging from standalone appliances to software implementations for consumer grade hardware such as smartphones and tablets. Despite the fact that OCR as a technology has matured for over a century, some fundamental problems remain unresolved which limits the practical use and adoption due to the lack of satisfactory accuracy and consistency. First and foremost the effectiveness of OCR relies on the quality of the source data, the raw image. If the raw data contains distortion, this can cause limited accuracy which in turn will cause faulty results. The process of taking a picture of text using a camera, or digitalizing a whole text document using a scanner will introduce interference and act as noise in the digital representation. Even though this kind of noise does not cause humans any problems to understand the semantics in the digital representation, it can cause issues for a computer trying to extract the text from the image.

Even if the source data is perfect, the diversity of different types of text font faces used has implications to the accuracy of OCR technology. In order to illustrate this, I conducted a small experiment. On the sample computer, running Ubuntu Linux (14.04 LTS), 540 different font faces were present. 462 of these are able to represent the English characters. Using the “ImageMagick”⁸ library and some bash-fu⁹, we created 462 images containing the English sentence “*The cake is a spoon.*”, each image using

⁸ImageMagick - <http://www.imagemagick.org/script/index.php>

⁹bash-fu - <http://nb.urbandictionary.com/define.php?term=bash-fu>

a unique font face. I then attempted to extract the text back from these images using the free, open source OCR engine *Tesseract OCR*, using the default configuration. *Tesseract* was chosen because we were already aware of an implementation of this engine for the Android platform. In this way, these experiments *could* be relevant for the implementation of the OCR functionality in the OffPAD application, which is described in section 5.2 on page 71.

In the experiments I only differentiate between successful and unsuccessful attempts, where a successful attempt returned the sentence with a 100% accuracy. This means that each letter in the OCR result is identical to the corresponding letter in the source string. For example, an OCR result of "The cake is a spoon." is considered successful, whereas a result of "THE CAKE IS A SPOON." is considered unsuccessful.

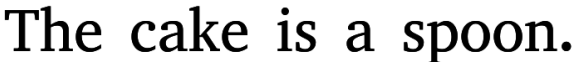

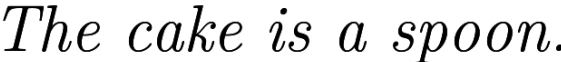


In our experiment the correct string, "The cake is a spoon.", was extracted successfully from 93,50% of the images. From the remaining 6,49% of the images, the extracted text was less than 100% accurate, or completely incorrect, and was considered unsuccessful. A small sample of these images, alongside with the text that was extracted can be seen in table 3.2.

In the first part of the experiment, the images used were quite suitable for OCR, since they do not contain any distortion and the text is perfectly horizontally aligned within the image. I was curious to see how the OCR processing would perform when the source images were suboptimal. First I wanted to see how the horizontal alignment of the text would affect the OCR processing. This would simulate a scenario where one is taking a picture of some text, without holding the camera in perfect alignment with the text. In order to test this, new images were created where the text was rotated 1°, 2°, 3°, 5° and 10°, for all the same fonts which were used in the first part. In total 2310 new images were created, 463 for every degree. Refer to table 3.3 to see how the rotation affected the image.

The OCR extraction test used in Part I of the experiment was repeated for the images where the text had been rotated horizontally. In the results from the second part of the experiment, I found that horizontal rotation does in fact affect the accuracy of the OCR process. With only 3° horizontal rotation, the OCR software was unable to extract the correct text for over 50% of the images. When applying a rotation of 10°, I was only able to achieve a successful extraction from one out of the images. The results from part II of the OCR experiment can be seen in figure 3.3. Part II of the experiment shows that horizontal rotation of the text greatly reduces the number of font faces suited for OCR, and affect OCR accuracy.

For the third part of the experiment, I wanted to see how vertical rotation of the text would impact OCR performance. This would simulate a scenario where one is tilting the camera when taking a picture of some text. This is illustrated in figure 3.4. In the top of this figure, we see what the text looks like without any tilt. In the center of the figure, we see how the text is displayed when a 45° tilt has been applied to the same text. I created new images where I applied 0°, 15°, 45° and 60° tilt. In total 1848 images were created. The same OCR test used in part I and part II of this

Table 3.2: OCR experiment - Part I

Extracted text	Image
The cake is a spoon.	
"I0 cake is a SIIIIIII.	
The cake 2'8 a spoon.	
Thecake'waopoon.	
The cake is a spam.	

experiment was conducted on these images.

As seen in the results, vertical rotation did not affect the OCR processing in the same degree as the horizontal rotation did. However, a gradual performance drop is seen when the degrees of rotation increase.

These experiments have little scientific value due to the way they were conducted. For instance, the results from the OCR processing were never cross-checked against what a human would interpret from the images. However, the results do at least indicate that OCR processing (at least not with the OCR software that was used during the experiments) is as of today an imperfect technology. The quality of the source data has an impact on reliability, and the simulations indicate that the horizontal or vertical tilt does affect the performance of the OCR processing. That being said, since only one OCR engine was tested, it might well be that other OCR software libraries would perform better in similar experiments.

While conducting these experiments, another key observation was made. When acquiring a picture for a computer display, there was a huge difference in the amount of visual noise which was introduced in

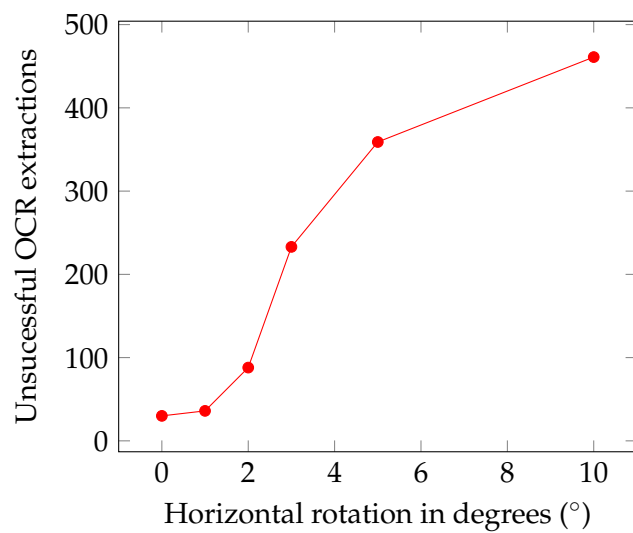


Figure 3.3: Results from OCR experiment - Part II

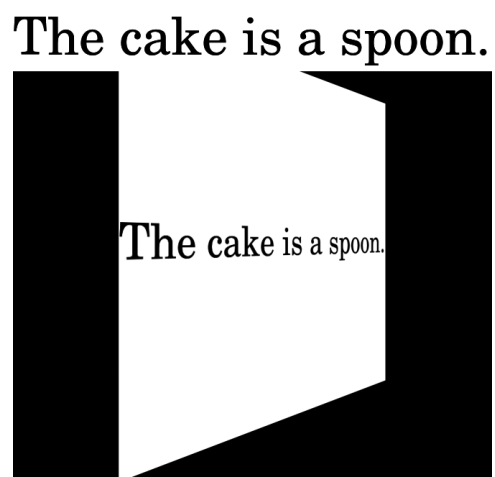


Figure 3.4: Text displayed with and without 45 °pan

Table 3.3: OCR experiment - Part II - Horizontal text rotation

Horizontal rotation	Image
0°	The cake is a spoon.
1°	The cake is a spoon.
2°	The cake is a spoon.
3°	The cake is a spoon.
5°	The cake is a spoon.
10°	The cake is a spoon.

the image depending on display *type*. It was attempted to acquire images from two types of computer displays: LCD (Liquid crystal display) with matte coating and LCD with glossy coating. These types of LCD screens are commonly used in both laptops and workstations. The advantage with LCDs with glossy coating is that color and contrast appear more vivid, however a major drawback is that this type of display reflect more light than its matte counterpart [12]. This causes a major problem when taking an image of the display using a camera, as this reflected light introduce visual noise. This is illustrated in figure 3.6. In the left side of this figure, a picture is taken of a matte display, whereas on the right side it is acquired from one with a glossy coating. It is necessary to point out that the light conditions were not identical when the two pictures were taken. The matte display could reflect more light when used in the light conditions used for the glossy display, just as the glossy display could reflect less light when used in the light conditions used for the matte display. In other words, this illustration is only used to emphasize the problem.

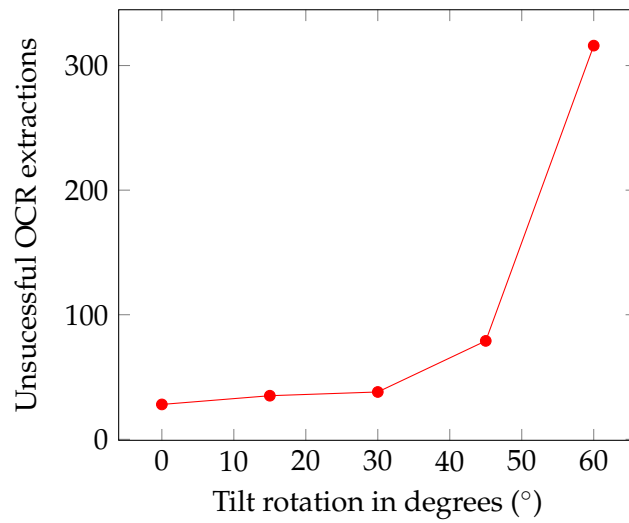


Figure 3.5: Results from OCR experiment - Part III

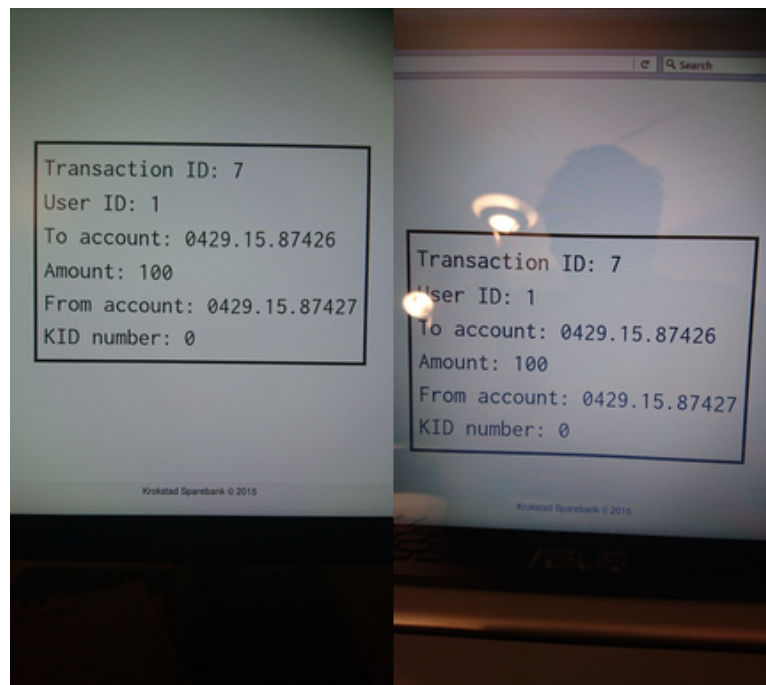


Figure 3.6: Difference in reflection between matte LCD and glossy LCD

In addition to *Tesseract*, there are several different OCR software alternatives available. Some are proprietary, closed source products such as ABBYY FineReader¹⁰ and Transym TORC¹¹. Others are open source projects, with less restrictive licensing developed by volunteers from across the globe. The “Tesseract OCR” project is one of the latter, however, it was originally developed and maintained by HP Inc. from 1984 to 1994. By the end of 1994, development ceased completely, and it wasn’t until 2005 the code base was released as open source for the public [28]. As a consequence, several forks¹² of the project appeared, including “*tess-two*” which is an implementation of the Tesseract OCR engine for the Android platform.

¹⁰ABBYY FineReader - <http://www.abbyy.com/>

¹¹Transym TOCR -<http://www.transym.com/>

¹²A software fork is the term describing a software project which is a result of copying the complete code base from an existing project, and then is developed independently from the original project

Part II

**OCR Based Data
Authentication**

Chapter 4

Design

The prototype was designed based on the use case provided by the OffPAD project. In this use case, users are able to authenticate online banking transactions made on any (possibly infected) computing device by using a separate hardware device (from now on only referred to as *the OffPAD*) in a secure manner. The basis for the use case is a specific attack scenario where a user is using a compromised client terminal (computer) when conducting online banking activity, as seen in figure 4.1. For this use case, it is assumed

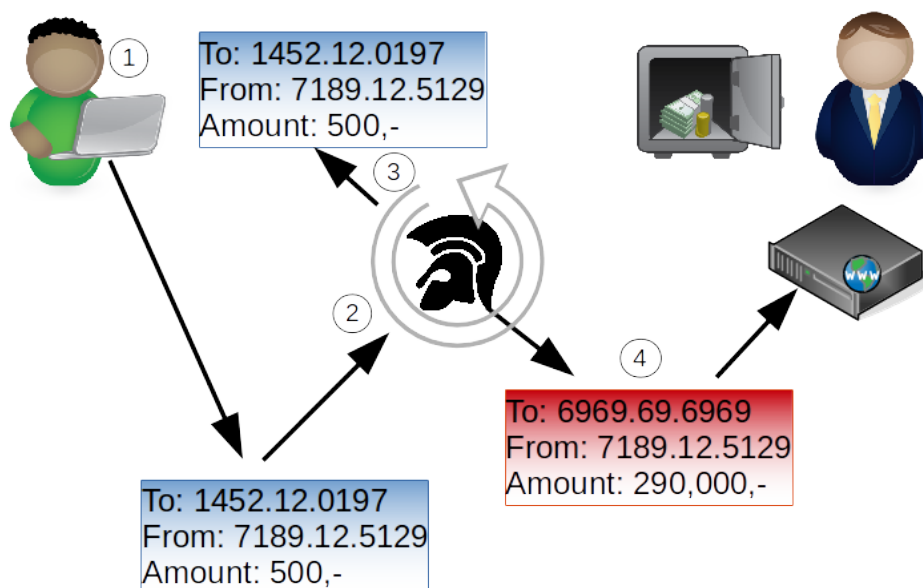


Figure 4.1: Attack scenario

that an OffPAD has already been provisioned to the user, and has been initialized so that the device itself is able to conduct owner authentication of the user based on a biometric modality, such as a fingerprint or iris recognition. It is assumed that the service provider (in this scenario, the bank) is aware of some unique cryptographic element found inside the specific OffPAD, such as the public key in an asymmetric key pair. As a part of the use case, a user will use a web-based online banking service

Nr.	Message description
1.	User types the transaction data in a browser window on the client computer
2.	The transaction data is intercepted by a trojan within the client computer
3.	The trojan displays the transaction data provided by the user on the client computer display
4.	The trojan alters the transaction data before sending it to the online bank

Table 4.1: Sequence of messages and actions in the attack scenario

to register a banking transaction with a client terminal. It is assumed that the terminal is infected by malicious software (from now on referred to as *malware*), which is capable of altering the transaction provided by the user without the users' knowledge. This alteration occurs inside the web browser of the infected terminal, and is known as a Man-in-the-Browser attack. While different online banking systems operate differently, it is common that the user is displayed the transaction that has been entered in the browser window, in order for the user to confirm the transaction before it is processed within the banking system. It is assumed that the online banking service uses this method to authenticate transactions. While the transaction is being displayed on the computer screen and is confirmed by the user, the malware crafts and sends a different, arbitrary transaction towards the bank, as seen in figure 4.1 (2), (3) and (4).

As a means to mitigate this attack, it has been proposed to use the OffPAD as a security measure by using it to authenticate the transaction. In order to transfer the transaction details from the assumed insecure device to the separate secure hardware device, a picture is taken of the computer's display. By using optical character recognition (**OCR**) the details are extracted from one device to the other. Please refer to section 3.2 on page 42 for a general description of optical character recognition. This is intended to provide, what we call, the "what you see is what you sign" (WYSIWYS) property.

After the transaction details have securely been transferred to the OffPAD, the user can then verify the transaction since it is assumed that we can trust the OffPAD to display the authentic data (what has been displayed on the screen). If the user finds the data displayed valid, it can confirm or authenticate the transaction by providing a biometric identifier, such as a valid fingerprint. After being authenticated by the user on the secure device, the authentication is sent towards the bank either via the insecure client device, or via another means of data transportation. Since the OffPAD is capable of performing cryptographic operations, the authentication itself is secured from any MitM attacks. In the document *Deliverable 5.2A* (please refer to Appendix A on page 101) the OffPAD project provides a high level description of how the OffPAD is utilized in order to authenticate data in such a use case. This ceremony is further

described in table 4.2 together with figure 4.2.

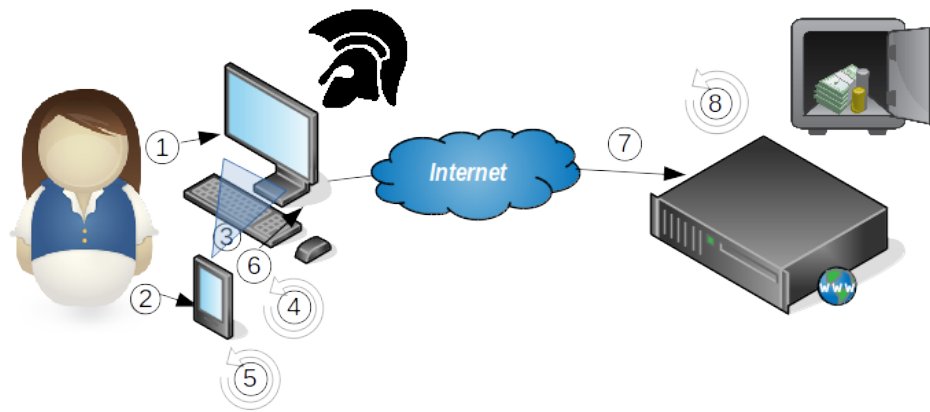


Figure 4.2: Ceremony using the OffPAD for data authentication in an online banking use case

Nr.	Message description
1.	User types the transaction data in a browser window on the client computer
2.	User activates the OffPAD to take a snapshot of the browser window
3.	Snapshot is taken of the text displayed in the browser window on the VDU
4.	The OCR function recovers the transaction data from the snapshot
5.	MAC generation with the transaction data and the user-password as input
6.	OffPAD sends the MAC to the client computer
7.	Client computer sends transaction data together with MAC to server
8.	Server verifies that the MAC corresponds to the received transaction data

Table 4.2: Sequence of messages and actions for data authentication ceremony

In the use case three main components have been identified: the client machine, the OffPAD and the online banking web application. These components can be divided into two groups: 1) the client side (the user) and 2) the server side (the service provider's systems). This is visualised in figure 4.3. In order to create a realistic test scenario for any usability testing, the prototype must implement both the client side and the server side. For the server side we only provide a mock-up of an online banking which includes the needed software for our ceremony. Any realistic server-side application would need this software module.

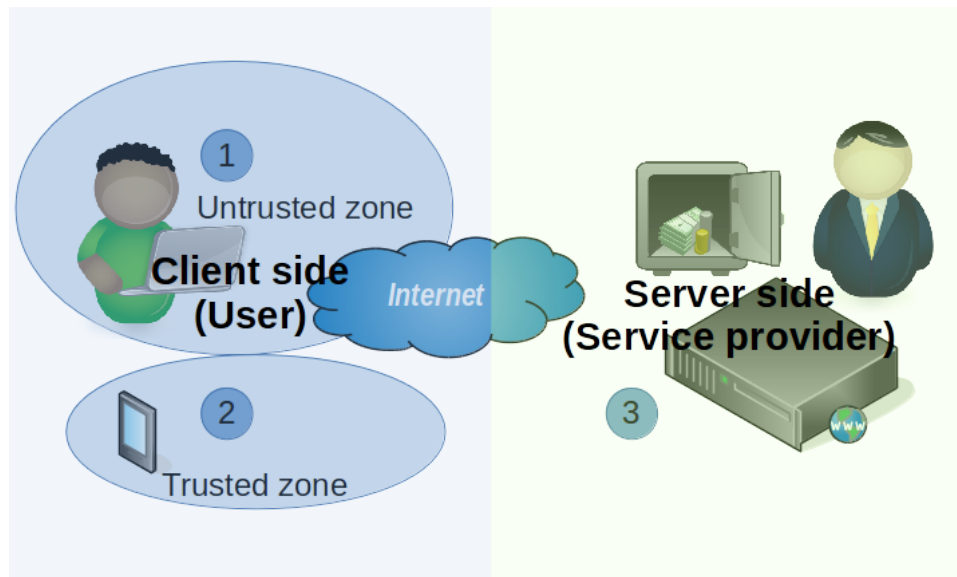


Figure 4.3: Prototype design overview

4.1 Client side

As previously noted, the client side of the prototype consists of both the OffPAD itself and the web application present on the user's terminal. During the OffPAD project, there have been several proposals to what the OffPAD should actually be. At first it was discussed to implement the OffPAD as a lightweight smart card. Later it was discussed to implement the OffPAD as either a smartphone or a tablet. A high fidelity prototype of an Android tablet and smart phone version was produced in small numbers. Before the project was put on hold in January 2016, it was decided that the OffPAD was to be implemented as a smartphone backcover. The backcover was to communicate with the user's smartphone using USB, as seen in figure 4.4. This formfactor is still used in the continued OffPAD project from October 2016.

For the communication between the OffPAD backcover and the smartphone, several technologies were considered. Wireless technologies such as **NFC** (Near field communication), **Bluetooth** and **WI-FI** were seemingly good candidates. These are all very mature, tried and tested technologies, which are widely adopted. However, common to all wireless technologies, these are also subject to over-the-air attacks (OAT). OAT attacks are becoming more common, and several severe attacks has been presented lately [11]. If the OffPAD were to implement a wireless technology, and there is found a vulnerability in this technology in the future, an attacker could potentially compromise the device with only limited physical proximity to it. Considering this, the project felt that a wired technology such as USB would be a better alternative. By using a wired technology an attacker would have to be in very close physical proximity to the device in order to attack it. This would require a physical cable between the devices, which

could impact how users perceived it in terms of usability. However, since the backcover was to be attached to the smartphone anyway, the project felt that having them interconnect using a short physical cable would have little implication on the usability of the device.

The smartphone (figure 4.4 ①) would provide means for acquiring images of the data which is to be authenticated and means for communicating the resulting signature back towards the online banking system. This could be done either by relaying the information back to the client machine, or directly using the phones inbuilt WWLAN or WI-FI capabilities. OCR, being both a CPU and memory intensive process is also to be conducted on the smartphone. In this way the CPU and memory requirement for the OffPAD were dramatically reduced, compared to a scenario where the OffPAD itself was to perform OCR on a raw image provided by the phone. This also meant that the OffPAD could be produced at a much lower cost in addition to reduced power footprint. It is assumed that the cost per device will impact the degree of adoption of this technology by service providers. The secure OffPAD backcover (figure 4.4 ②) should only be responsible for displaying the user the data which is to be authenticated in a secure environment and perform the cryptographic operations on the data.



Figure 4.4: Smartphone with OffPAD secure backcover

Since the project came to a halt before the final design of the OffPAD backcover was ready, there was only a general idea of what it would look like and the features it would hold. A simplified sketch of the OffPAD backcover can be seen in figure 4.5. The OffPAD backcover was to have a small, low power CPU and a modest amount of memory. It would be able to conduct cryptographic operations such as digital signing, hashing and verification of digital signatures. Each device would have a unique digital certificate and corresponding key stored in non-volatile memory. The work on the OffPAD backcover has been resumed in October 2016.

The project planned to implement the display as a small e-ink screen (figure 4.5 ①). This was due to the low power consumption compared to other display technologies, which would improve battery life. There was a general consensus within the project group that this could improve the overall usability of the device.

Figure 4.5 ② refers to the integrated finger print scanner. The idea was that in order to authenticate any data using the OffPAD, the user

itself would have to authenticate using this finger print scanner together with a PIN. We call this “owner authentication”. In this way the OffPAD would provide three factor authentication towards any service provider as it would fulfil the three requirements described in Section 2.1.1 on page 12:

1. Something the user has (the OffPAD itself and its unique certificate and corresponding key)
2. Something the user knows (the PIN)
3. Something the user is (fingerprint/biometric modality)

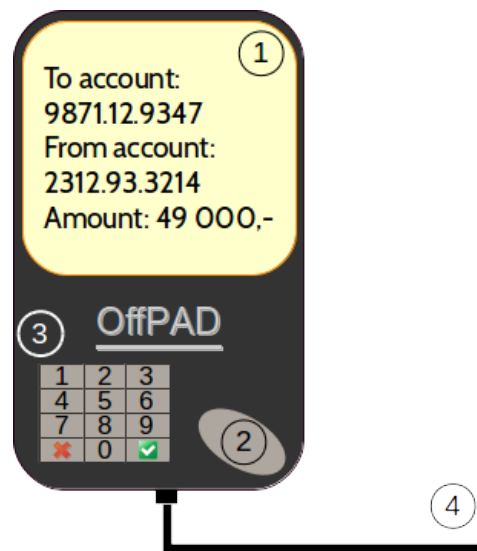


Figure 4.5: OffPAD backcover

As seen in figure 4.5 (3), the OffPAD also comes with a small keypad for user interaction, such as when the user would have to enter the PIN to authenticate a transaction.

Figure 4.5 (4) refers to the μ USB cable which interconnects the OffPAD and the smartphone it is being used together with.

At the time of writing, no high fidelity prototype of the backcover has been produced. A proof-of-concept prototype was developed in very limited numbers, and was not available for this Master’s project.

Since the physical manifestation of the OffPAD was somewhat unknown during the early phases of our work, a decision was made to implement the OffPAD as a software application for the Android smartphone platform. The application simulates the cryptographic operations the OffPAD would conduct if it were to be produced. The smartphone’s display would be used to interact with the user, as opposed to a separate e-ink display. The application will also have to implement OCR capabilities in order to extract the transaction data from an acquired picture of the transaction. In this way, it was possible to showcase some of the features proposed for the OffPAD, without having a real prototype of the product at

hand. This however, will potentially create an unrealistic user experience during any usability tests, and therefore could limit the transferability of the results derived from these tests.

The second component of the client side is the online banking web application which is running in the user's web browser. This component is from now on referred to as the "*front-end*". This web application will initially be delivered to the user's machine from the bank's web server. As soon as it is downloaded, it is assumed that it can be tampered with by any malicious software present on the user's machine. In order to create a realistic user setting, the web application must at least implement the following features:

- Let users register transactions
- Let users view the current balance
- Let users view the transaction history
- Let users view authentication attempts of previously registered transactions

4.2 Server side

In the prototype, the server side back-end will mimic the logic of a simple online banking system. The back-end is responsible for making changes to the account data stored in the database, based on the requests coming from the client web application and the OffPAD client. In other words, the back-end consists of both the software running on the web server, as well as the database itself. The back-end must therefore support the following functionality:

- Maintain a database containing information regarding users, accounts, transactions and authentication attempts
- Validate transaction data from the web application
- Receive transaction details and signature from the OffPAD application
- Verify the authenticity of the data received from the OffPAD application using the signature

Chapter 5

Implementation

After identifying the main requirements for the prototype based on discussions within the OffPAD project and literature provided by the group, the phase of development began. The development process can be said to be based on what is known as the “waterfall model” in software engineering [29, p. 29-32]. This process is illustrated in figure 5.1.

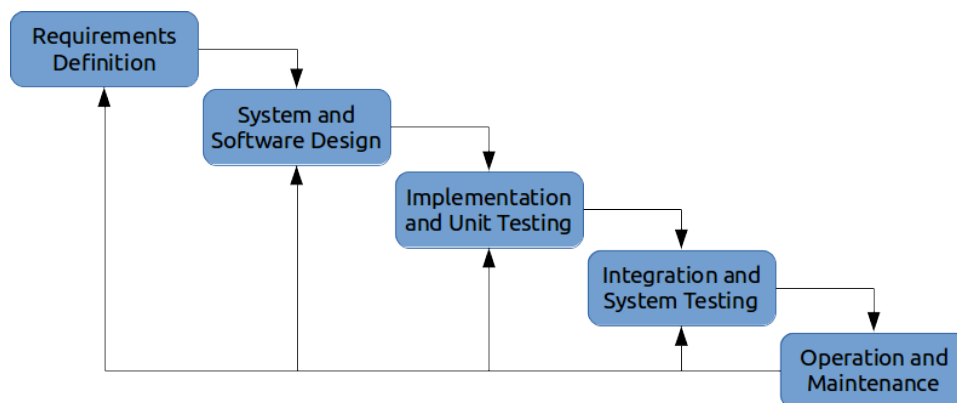


Figure 5.1: Waterfall Model for Software Development

Prior to writing any actual code, the requirements were gathered, and the design for the prototype was developed, as described in the previous chapter (chapter 4 on page 53). These activities are denoted as the two first steps in the waterfall model: “*Requirement Definition*” and “*System and Software Design*”. This also involved deciding on the technologies which would be most suited for the implementation.

During the next stage, “*Implementation and Unit Testing*”, the separate parts of the prototype were developed separately. As previously described, the prototype can be divided into the following components: the front-end, the back-end and the OffPAD Android application. In addition to these three components, the database was treated as an individual component throughout the development process. The front-end and back-end together constitute the web application, in consequence they were treated as one component throughout the development phase. Since both the web application and the OffPAD Android application utilize the database, this

was implemented first. Following the database, development of the web application begun. When both the database and the web application were operational, development of the OffPAD Android application was started.

When all of the components were finalized, the *“Integration and System Testing”* phase begun. Testing the individual components together as a complete system identified previously unknown flaws and bugs in the individual components which had to be corrected. In essence, this led me back to a new iteration of the *“Implementation and Unit Testing”* phase. Some flaws also required alterations in the database design. This led development back to the *“System and Software Design”* phase.

At the time of writing, the status of the prototype is in the *“Integration and System Testing”* phase. It was used in a pilot usability study (please refer to chapter 6 on page 85). This pilot study reveals several flaws in both design and implementation of the prototype, which should be corrected.

5.1 Web application

The web application portion of the prototype was developed to run on a LAMP¹ stack running in a virtual machine. The operating system used for the virtual machine was Debian 7, which was the latest stable release of Debian² when the development began. Virtualization was achieved using Oracle VirtualBox³. This virtual machine acts as the web server and database server for the prototype. This virtual machine was also configured to communicate with the smartphone running the OffPAD Android application using a USB WiFi dongle.

The back end consists of a relational database instance (MySQL), and an application written in PHP for handling the business logic. The front-end, the web application being served to the user’s browser was created using HTML, CSS and JavaScript.

5.1.1 Database design

I began designing the database schema for the web application, based on the identified requirements. A brief overview over the database as a whole can be seen in figure 5.2. Data types and related configurations can be seen in figure 5.3. Both diagrams were generated using Schemaspy⁴. Some columns have been added since the first draft of this design, and others have been modified during the course of development.

According to this design, information about every user is stored in the “users” table. Each user is identified by a unique *UserID*, which acts as the primary key for the table. Other information such as each user’s username, first and last name and e-mail address is also stored. Passwords are not stored in plain text in the database, but rather the hash sum of each

¹LAMP - Linux, Apache, MySQL and PHP

²Debian Linux - <https://www.debian.org/>

³Oracle VirtualBox - <https://www.virtualbox.org/>

⁴Schemaspy - <http://schemaspy.sourceforge.net/>

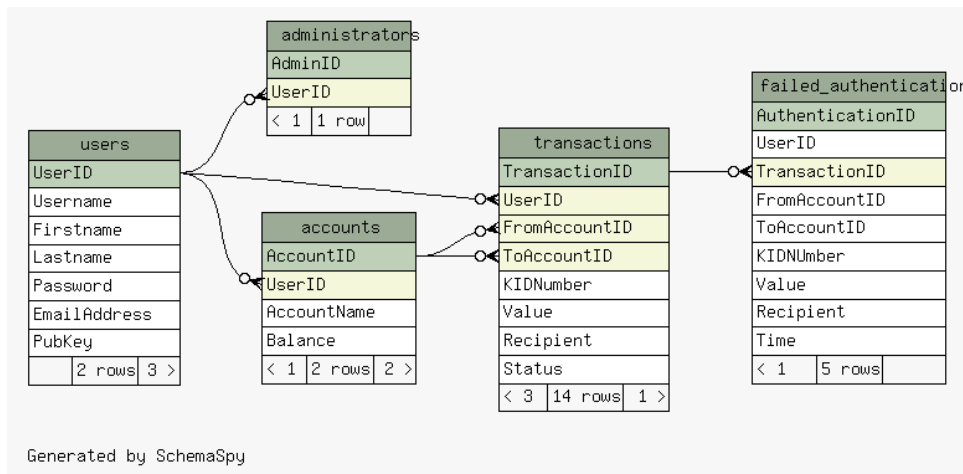


Figure 5.2: Database design overview

Table	Column	Type	Size	Nulls	Auto	Default
accounts	AccountID	int	10		√	
accounts	AccountName	text	65535	√		null
accounts	Balance	double	22	√		null
accounts	UserID	int	10	√		null
administrators	AdminID	int	10		√	
administrators	UserID	int	10	√		null
failed_authentications	AuthenticationID	int	10		√	
failed_authentications	FromAccountID	text	65535	√		null
failed_authentications	KIDNumber	text	65535	√		null
failed_authentications	Recipient	text	65535	√		null
failed_authentications	Time	timestamp	19			CURRENT_TIMESTAMP
failed_authentications	ToAccountID	text	65535	√		null
failed_authentications	TransactionID	int	10	√		null
failed_authentications	UserID	int	10	√		null
failed_authentications	Value	double	22	√		null
transactions	FromAccountID	int	10	√		null
transactions	KIDNumber	text	65535	√		null
transactions	Recipient	text	65535	√		null
transactions	Status	enum	13			
transactions	ToAccountID	int	10	√		null
transactions	TransactionID	int	10		√	
transactions	UserID	int	10	√		null
transactions	Value	double	22	√		null
users	EmailAddress	varchar	255			
users	Firstname	varchar	65			
users	Lastname	varchar	65			
users	Password	varchar	32			
users	PubKey	blob	65535	√		null
users	UserID	int	10		√	
users	Username	varchar	65			

Generated by SchemaSpy

Legend: SOURCEFORGE.NET

- Primary key columns
- Columns with indexes

Figure 5.3: Table and column overview

password is stored. The *Pubkey* column contains the user's public key. Public keys are X.509 certificates stored using the *binary blob* data type.

The "administrators" table contains reference to the registered users (users found in the "users" table) which also have administrative rights within the web application. The users listed here are granted special permissions, such as viewing or modifying the server's key pair. Being able to have this feature is not a necessity for the prototype itself, but it was

a helpful feature during the development process.

Each registered user disposes $0 \dots n$ banking accounts listed in the “accounts” table. Every account has a unique account identifier (*AccountID*). The monetary value tied to each account is represented by the *Balance* column. The user disposing the account is denoted by the *UserID* column. The idea behind the *AccountName* field was originally to allow users to add names to their accounts, i.e., “Savings” or “Household”. During the course of development it was decided that it would be more realistic in a user test scenario if each account was assigned an arbitrary real-life identifier used by banks. Due to this, this field is in fact used to store the fictive account number, i.e., the number used to identify a given bank account. The format for this identifier varies from country to country, e.g., in Norway an eleven-digit identifier using the following format has been used since 1967: *xxxx.yy.zzzzz*[4]. This allowed an account to be referenced as “0112.18.97812”, as an example.

Each banking transaction is registered in the “transactions” table. Each transaction is identified by a unique transaction ID (*TransactionID*). The user ID of the user requesting the transaction is also stored with the ID for the source and destination account. The *Value* column is used to specify the amount of money which is to be transferred from the source account to the destination account. The *Status* field is used to denote whether a transaction has been processed by the back-end or not. The idea was that once a transaction has been requested by a user, the transaction would get a “Requested” status, and it would not be processed until the transaction was authenticated using the OffPAD Android application. When the back-end received this authentication, and verified it as authentic, it would change the status to “Authenticated”, before any processing would occur.

In order to store information about any failed authentication attempts a separate table, “failed_authentications” was implemented. This table is similar to the “transactions” table, but has an additional column for saving the time stamp for when a failed authentication attempt was made. This allows users to review any failed authentication attempts.

5.1.2 Back-end & Front-end

The web application as a whole consist of what is usually called a *back-end* and a *front-end*. The front-end is what is visible to the user while using the web application. Typically that would be anything which is processed and displayed by the user’s browser. In this way, the front-end acts like a *front* for the web application. Generally, a front end is built using technologies such as HTML⁵ (HyperText Markup Language), JavaScript⁶ and CSS⁷ (Cascading Style Sheets). The server will create and deliver documents crafted using these technologies to the the web browser which

⁵HTML - <https://www.w3.org/html/>

⁶JavaScript - https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

⁷CSS - <https://www.w3.org/Style/CSS/>

will do the job of processing the information found in these documents and render the web page, which is then visible in the browser display.

The back-end however, can be described as the part of the system the user cannot see, which handles the inner workings of the web application. Imagine the following scenario: a user visits a given web page. The server delivers an HTML document which instructs the web browser to render a page with an input field for the user's first name and a "Submit" button. The HTML document, now being displayed in the user's browser is now the front-end for the web application being served. Now, if the user enters his or her name into the input field, and presses the "Submit" button, a new request is made to the web server. This request now contains the name which was provided by the user. The code on the server-side which handles this request can be considered to be the back-end. This code could be written in i.e., Ruby, Python, Java or PHP, which all are popular server-side (also known as back-end) languages in web development. For example, this code could take the provided name and store it in a database, or check its popularity against some other database. The key point is that all of this is happening without being visible to the user. Let us imagine that the code handling this specific request only takes the name, and adds a prefix somewhere along the lines of "Greeting, ". Then constructs a *new* HTML document containing this prefix concatenated with the name provided which is sent back to the user's browser. This document is, as before, rendered by the browser and a page greeting the user is displayed in the browser window. Now this page can be considered to be the front-end of the web application. Now, this makes it clear that a web application can (but does not have to) consist of both a front-end and a back-end, which interact using HTTP requests/responses.

Henceforth during this section, *web application*, *front-end* and *back-end* are used interchangeably. When the term web application is used, it refers to the application as a whole, in other words the front-end and back-end combined. The term front-end will refer to the UI (User Interface) provided to the user through his or her browser. The back-end refers to any code invisible to the user. The complete source code tree can be found in online using the following link: www.kaffemarius.com/Web_Application.tar.gz. If prompted for a username and password, *masterthesis* and *masterthesis2016* should be provided.

For the back-end, the PHP⁸ (PHP: Hypertext Processor) language was chosen due to its gradual learning curve, which seemed like a good alternative for those who have little or no experience with web programming. The version of the PHP engine used throughout development was PHP 5.4.36-0+deb7u3. As for the technologies for the front-end the de facto standards, HTML, CSS and JavaScript were all used. A third party front-end framework, Bootstrap⁹ (version 3.3.2), was used as a helpful tool when implementing the design.

The name "Krokstad Sparebank" was chosen for the mock-up bank,

⁸PHP - <https://secure.php.net/>

⁹Bootstrap - <http://getbootstrap.com/>

therefore the URL `www.krokstadsparebank.no` is referenced both throughout the text as well as in some of the figures provided. The prototype was configured in such a way that the domain `krokstadsparebank.no` points to the IP address of our virtual web server which is serving the actual web application.

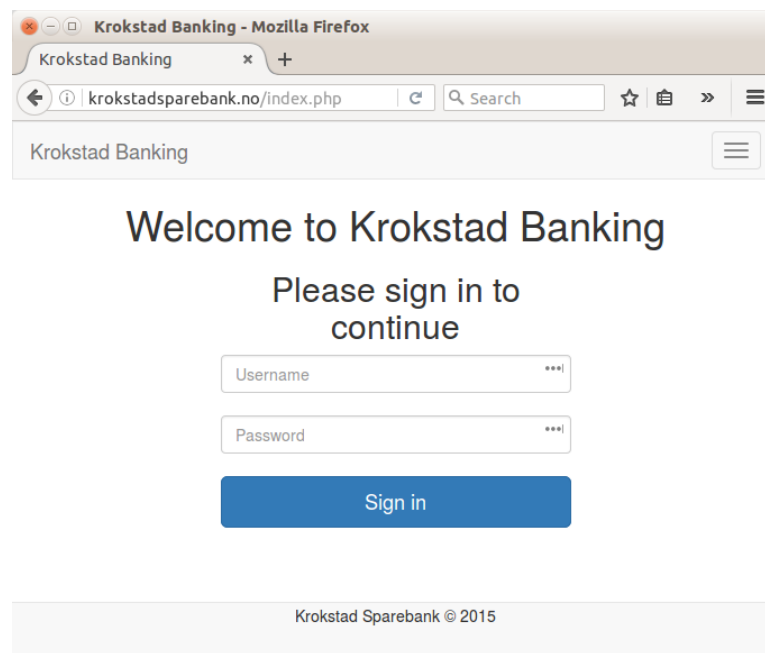


Figure 5.4: Web application - Login page

The front-end of the web application offers a simple login screen when first loaded. In this way, a user is able to log in using his or her username and password, as illustrated in figure 5.4. For this prototype we found that a username & password scheme was sufficient, and multi-factor user authentication was not implemented, which would be recommended for a real-life online banking system. Any user will have to be registered in advance, before attempting to log in. When a user has logged in, she gains access to more functionality, such as registering new transactions, review account balance and transaction history, and review transaction logs. As seen in figure 5.5. In addition, the user is able to upload her public key, which is required for the web application to validate the digital signatures used for the authentication of the transactions. It is required that the certificate provided by the user is formatted as an X.509 certificate. This is illustrated in figure 5.6.

The left side of figure 5.7 illustrates the page used for registering new transactions, whereas the right side of this figure illustrates the transaction details page the user is redirected to whenever a new transaction is submitted. This page, the transaction details page, is the page the user would capture an image from using a smart phone for authentication using the OffPAD Android application. When a new transaction is registered within the backend, it will, as previously mentioned, be put

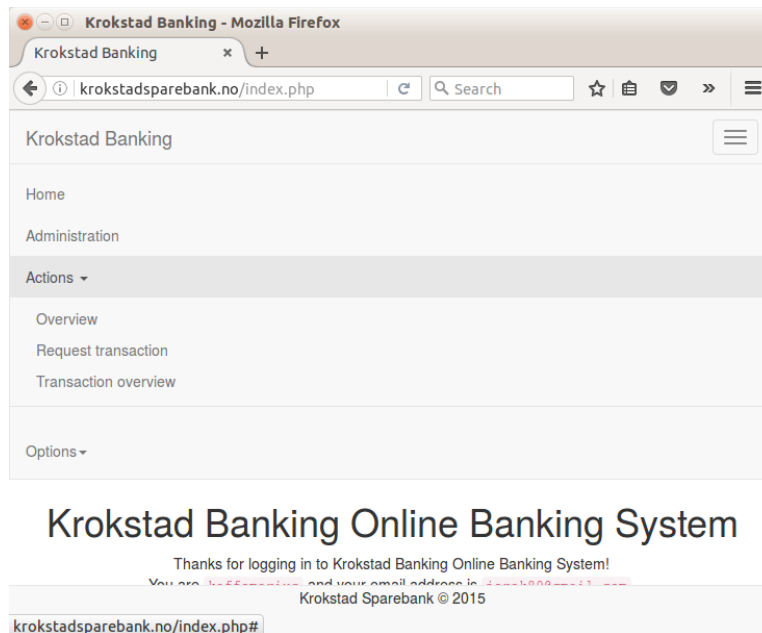


Figure 5.5: Web application - Functionality when logged in

User information

Here you can review your user information.

Username: kaffemarius
First name: Marius
Surname: Haugen
E-mail: jorek89@gmail.com
Certificate:

-----BEGIN CERTIFICATE-----

Update public key

Certificate data:

Public key:

Key length: 4096 bits

Key:

```
-----BEGIN PUBLIC KEY----- MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICGKCAgEAlt6V7BIW+P28dM83gS1c
Yg59On6Izvm8E6w74RcNWWroegz3mZE3+JYG+c/sBry1fhidh9Dd+yTfKp/Fbn6l
4n4Elo4f85CqKikXrtPlPf8OVEXfQ6CIKpFyRE3nloR0e/+ULTcBVyi2oufOPwx+
3fL49aaPPzqJrbswMMwzVU2TM6kpBHwmBKwqoPsNEhkaWEGClY4WcVJX505KT7gBF QugvvrFasyB/3LjPpfjTQJHZgktj
/dDcGiVdI/4WPMvoJPb8WthlBXHOijwSVJZw3
l1DxFD7BgrvDYXqFhY0QuLj0uP/lzuiPzNyJy7URI6amILeAQ7XZUrHr3i7xsjU1
aMP5Lny0gxbVgTKVLCV7gRbKC+TxD2DIgfWAE9dtDnySaGY6JJXbMq55rNU7Ra7a
4S9mfhRSLTWKOCpZvr9x8I27m4W+irSp1SXYWWu/tAp8Xy7elz1m3dLZF4LwPxSL
r9Hg6T6Tlp+ALrUr56NBfjcmQCGXJzPFR04JEOPemn2++vFtFH06iDvk+biel5pP
LiLJl5GlyKRNldo5DJmhIsH5HSyxvHRnfAnX7LKNmzoDPsdqFAG0i0p4o3DjfPK2
C9tjH9hYV1LmCcsPgeGZI4uoUbu36Q+PjDVzaxy3Av6zpsAw+jZj3aAJLA8uS8XJ C/gXV9Zt/KIeSMTW/FF4I9MCAwEAAQ==
-----END PUBLIC KEY-----
```

Figure 5.6: Web application - User info

in a “pending” state. This means, in context of this web application, that the transaction will not be processed before the web application receives a valid authentication for the transaction in question. This, as we know, is

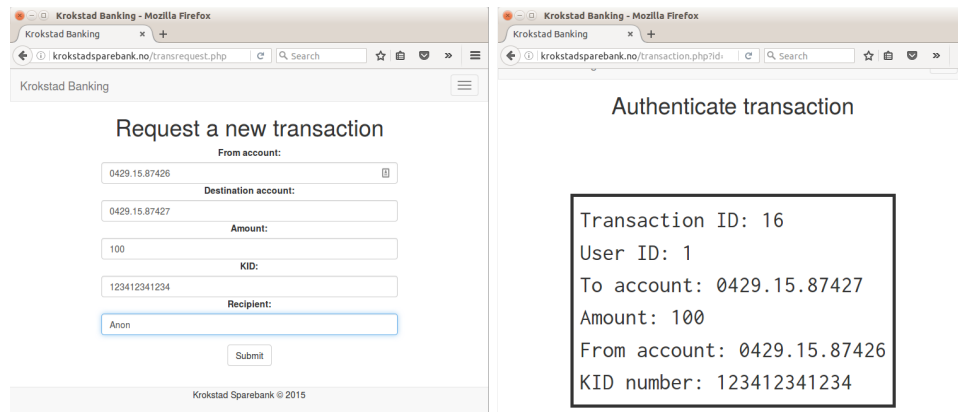


Figure 5.7: Web application - Transaction registration and transaction details

done using the OffPAD application running on an Android smartphone, and is described in depth in section 5.2 on page 71.

Any transaction authentication attempts towards the web application is handled by the code defined in the *authenticate.php* file. This process is initiated by a HTTP request towards the URL <http://www.krokstadsparebank.no/authenticate.php>, made from the OffPAD Android client. The PHP code expects an HTTP POST¹⁰ request, which includes a UserID, a TransactionID, the encrypted transaction as well as a digital signature of the transaction details, as seen in listing 5.1.

Listing 5.1: PHP code handling authentication attempts

```
if ($_SERVER['REQUEST_METHOD'] != 'POST') {
    http_response_code(FAILURE);
    echo "Unsupported request method.\n";
    write_to_log("Got non-POST request. Doing nothing");
}
else {
    // Request was of type POST. 4 paramaters must be set: UserID,
    // TransactionID, EncryptedOCR and SIG of EncryptedOCR.
    $userid = $_POST['userid'];
    $transid = $_POST['transactionid'];
    $transaction = base64_decode($_POST['transaction']);
    $signature = base64_decode($_POST['signature']);
    // Write to test file to disk, just for troubleshooting
    $file = fopen("/var/www/krokstadsparebank.no/logs/sig.tmp", "w");
    fwrite($file, $signature);
    fclose($file);
}
```

Since the transaction details are encrypted using the web server public key, the web application will decrypt the details using its private key, which is stored as a file on the server itself. When the transaction details are decrypted (and stored in memory as plain text), the server will verify that the corresponding signature is valid, and that it was produced using the user's private key (which we assume is safely stored within the OffPAD application itself). This is achieved in the *verify_transaction_integrity* function, defined in *authenticate.php*, which can be seen in listing 5.2.

¹⁰HTTP POST - <https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.5>

Listing 5.2: Function for verifying digital signature of transaction details

```
function verify_transaction_integrity($plaintext, $signature, $userid) {
    $pubkey = get_pubkey($userid);
    if(empty($pubkey)) {
        /* No pubkey found. Either because of invalid UserID, or that the
           user haven't uploaded any pubkey information */
        return array("ret" => false, "msg" => "Verification Failure.
           Invalid user ID, or empty pubkey.\n");
    }
    $ret = openssl_verify($plaintext, $signature, $pubkey, "sha512");
    if($ret == 1) {
        return array("ret" => true, "msg" => "Verified OK.\n");
    }
    else if($ret == 0) {
        return array("ret" => false, "msg" => "Verification Failure.
           Incorrect signature.\n");
    }
    else {
        return array("ret" => false, "msg" => "An error occured while the
           system tried to verify the integrity of the transaction.\n");
    }
}
```

As seen in the listing, the public key for the user who is attempting to authenticate the transaction is retrieved from the “Pubkey” column for the corresponding user in the “users” table in the database (please refer to section 5.1.1 on page 62 for the complete database design). If a key is found in the database, by using the OpenSSL library ¹¹, the provided signature is tested against the provided transaction details using the key. If the signature is found to be valid, the transaction is processed further by using the *process_authentication_request* function. However, if the signature is *not* found to be valid, the authentication attempt is aborted, while an error is sent back to the OffPAD application which sent the request.

In the *process_authentication_request* function the transaction the user has requested to authenticate is retrieved from the database by using the provided transaction ID. Since the transaction from the OffPAD application is sent in a predefined format (refer to section 5.2.3 on page 80), the data has to be parsed before further processing. When it has been parsed, every field in the transaction stored in the database, is compared to the corresponding field provided by the OffPAD application. If every field matches, then the transaction is authenticated. If there is any divergence between the fields, the transaction will *not* be authenticated, and an unsuccessful authentication attempt will be logged in the “Failed_authentications” table.

```
function process_authentication_request($userid, $transactionid,
    $transaction) {
    //1. Fetch the registered transaction from DB
    $trans_from_db = server_get_transaction($transactionid);
    //2. Parse $transaction
    $trans_from_request = transaction_str_to_array($transaction);
    if($trans_from_request['ret'] != true) return $trans_from_request;
    //3. Compare each field in the transaction from DB with every field in
    $transaction
    if((strcmp($trans_from_db['userid'], $trans_from_request['userid']) ==
        0) && (strcmp($trans_from_db['src_account'],
        $trans_from_request['src_account']) == 0) && (strcmp(
        $trans_from_db['dst_account'], $trans_from_request['
        dst_account']) == 0))
```

¹¹OpenSSL and PHP - <http://php.net/manual/en/book.openssl.php>

```

&& (strcmp($trans_from_db['kid'], $trans_from_request['kid']) ==
0) && (strcmp($trans_from_db['amount'], $trans_from_request['
amount']) == 0)) {
//4 If match; Mark the transaction as approved, add to the
authentication table and return true
$ret = server_mark_transaction_as_authenticated($transactionid
);
if($ret['ret'] == false) {
return $ret;
}
return array("ret" => true, "msg" => "Transaction
authenticated.\n");
}
// If NOT match; log the attempt to failed_authentication table and
return false
// Log attempt.
$tmp = log_authentication_attempt($trans_from_request);
if($tmp['ret'] != true) write_to_log("MySQL error:".$tmp['error']."\n
".$tmp['sql']);
return ["ret" => false, "msg" => "Mismatch.\n"];
}

```

When a transaction is authenticated, it is put in a “*authenticated*” state. In a real-life setting, that would mean that the transaction would be processed (value is transferred from the source account to the destination account). In this prototype, nothing actually happens after a transaction has been marked as “*authenticated*”, other than being marked.

Transaction overview							
Pending authentication							
Transaction ID	From Account	Recipient	To Account	Amount	KID number	Status	Authentication History
ID: 7	0429.15.87427		0429.15.87426	100	0	Requested	Log
ID: 8	0429.15.87427		0429.15.87426	100	0	Requested	Log
ID: 9	0429.15.87427		0429.15.87426	100	0	Requested	Log
ID: 10	0429.15.87427		0429.15.87426	100	0	Requested	Log
ID: 16	0429.15.87426	Anon	0429.15.87427	100	123412341234	Requested	Log
Authenticated							
Transaction ID	From Account	Recipient	To Account	Amount	KID number	Status	Authentication History
2	0429.15.87427		0429.15.87426	100	0	Authenticated	Log
3	0429.15.87427		0429.15.87426	100	0	Authenticated	Log
4	0429.15.87427		0429.15.87426	100	0	Authenticated	Log
5	0429.15.87427		0429.15.87426	100	0	Authenticated	Log
6	0429.15.87427		0429.15.87426	100	0	Authenticated	Log
11	0429.15.87427	Sijan Gurung	0429.15.87426	100	123123	Authenticated	Log
12	0429.15.87427	Audun J	0429.15.87427	150	123123123123123123	Authenticated	Log
13	0429.15.87427	Terje	0429.15.87426	200	123123123123123123	Authenticated	Log
14	0429.15.87427	Audun J	0429.15.87426	200	123123123123123123	Authenticated	Log
15	0429.15.87427	Marius	0429.15.87426	100	1234	Authenticated	Log

Figure 5.8: Web application - Transaction overview field

All transactions, both those that are registered but have not yet been authenticated as well as those who are, are visible in the “*Transaction Overview*” page (which can be found in the drop-down menu illustrated in figure 5.5). This page, the transaction overview page, can be seen in figure 5.8. Transactions listed in the “Pending authentication” table are those that still require a valid authentication. Right below, in the “Authenticated” table, those transactions which already have been authenticated are listed. As seen in the figure, an “Authentication Log” is available in the leftmost column in each table. When a user presses this column, a new page displaying the authentication history for the

transaction in question is loaded, as seen in figure 5.9. The top row displays the transaction which has been received from the web browser, marked in blue. The two rows below, outlined in red, are two authentication attempts where the transaction details do not match the ones that were received through the web browser. In this way a user can investigate why

Transaction received from web browser ▼

Transaction ID	From Account	Recipient	To Account	Amount	KID number	Timestamp
10	0429.15.87427		0429.15.87426	100	0	-
10	0429.15.87427		1234.34.84220	100	0	2015-10-07 15:05:45
10	0429.15.87427		0429.15.33396	100	0	2016-07-14 14:53:49

Transaction received from OffPAD VDA ▲

Figure 5.9: Web application - Authentication history

a given transaction fails to authenticate. One reason could be that there is some malware running inside the client computer, which is altering the transaction within the browser before it is sent to the bank, as described in chapter 4 on page 53. Another plausible explanation for why a transaction will not authenticate is that the OCR processing conducted using the OffPAD application (as described in section 5.2.2 on page 75) has failed to extract the correct information, and the user has failed to notice it before attempting to authenticate.

5.2 Android client

The Android application was developed using Android Studio¹², which is an IDE (Integrated Development Environment) for the Android platform. The version of the Android SDK (Software Development Kit) changed throughout the development process, but the final version of the application was compiled using the version 23 in combination with Android Studio version 2.0.

The application was originally tested and developed using a Sony Xperia Z3 Compact¹³. Due to a savage encounter with a bathroom floor a second phone, a Sony Xperia Z5 Compact¹⁴, was used in the later stages of the project. The version of the Android OS changed (naturally, due to software updates from the manufacturer) during the project, but the final version of the application was tested in a version 6.0.1 environment.

For the required OCR capabilities a third party software library, tess-two¹⁵ (version 5.1.0-53-g08b4d41), was used. This project is according to their own web site a fork of the Tesseract Tools for Android¹⁶ project, which in turn is a implementation of the Tesseract OCR engine for the Android platform.

¹²Android Studio - <https://developer.android.com/studio/index.html>

¹³Sony Xperia Z3 Compact - <http://www.sonymobile.com/us/products/phones/xperia-z3-compact/>

¹⁴Sony Xperia Z5 Compact - <http://www.sonymobile.com/global-en/products/phones/xperia-z5-compact/>

¹⁵tess-two - <https://github.com/rmtheis/tess-two>

¹⁶tesseract-android-tools - <https://git.io/vXUG1>

The tess-two software, just as tesseract, requires a training data file in order to perform OCR processing. A training data file contains language and font specific information which helps the software identify and extract the correct information. For the prototype, it was decided to use a stock, training data file for the English language published by the tesseract project¹⁷. This file was then transferred to the smart phone's internal storage.

Since it is assumed that the OffPAD client already has been initialized in a way so that it holds a cryptographic secret, an RSA key pair for the OffPAD client was generated using the openssl library on a separate computer. The private key was converted to a PKCS8 format, and transferred to the smart phone's internal storage, together with the server's public key formatted in the PEM format. The public key for the OffPAD client was added to the dummy user already registered in the web application.

Having to manually copy these three files, the training data file, server's public key and the client's private key, is impractical for any real use scenario. This information should be stored within the application itself for more granular control, as well as better usability. However, this would require a more advanced system (at least for management of the different keys used), which is outside the scope of this prototype.

Applications for the Android platform consist of one or more *activities*. In this context, one can think of an activity as a separate application being responsible for a small set of features, which can be invoked by other activities within the application. This is similar to a function or procedure in a traditional computer program written in i.e., *C* or *Lisp*. Much like functions, an activity can receive data when being invoked (just like a function can be fed with *arguments* or *parameters* when being called), and return data back to its caller when exiting (similar to a function's return value).

In this way, it is possible to structure the OffPAD application according to the different steps in the work flow e.g., one activity for taking a picture, one activity for the authentication towards the web server. Based on the work flow presented in the security ceremony provided by the OffPAD project (see figure 4.2 on page 55), the following possible activities were identified:

- Main activity (default screen)
- Acquire Picture activity
- Conduct OCR activity
- Verify Transaction Details activity
- Authenticate activity

¹⁷tesseract-ocr-3.01.eng.tar.gz
tesseract-ocr-3.01.eng.tar.gz

Based on the compiled list of possible activities which could be implemented in the application, sketches for each of the activities' UI (user interface) were produced. For this, the open source GUI (Graphical User Interface) prototyping software "Pencil"¹⁸ was used. An example of these sketches can be seen in figure 5.10. All the UI sketches produced during this stage can be seen in Appendix B.



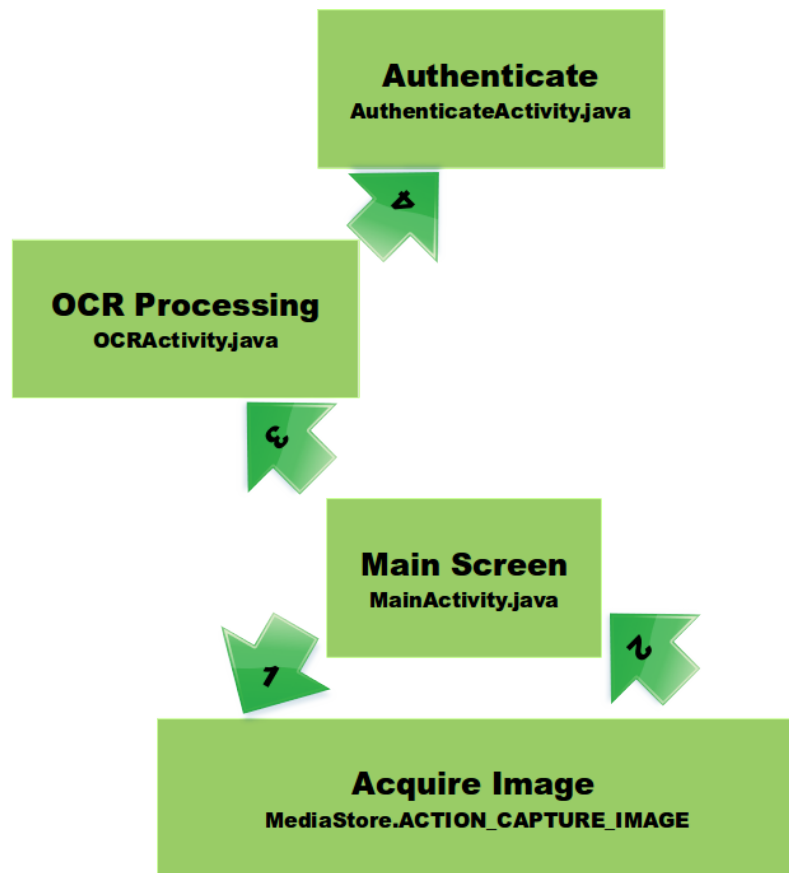
Figure 5.10: Application UI sketches - Part I

Based on these sketches, development of each activity began. It was early discovered that there was a built-in activity in the Android SDK¹⁹ for acquiring images using the camera, if present. Due to this, there was no need to create a new activity for acquiring images from scratch. It was also decided to combine two of the possible activities, the "Verify Transaction Details" activity and the "Authenticate" activity, to a single activity: "Authenticate Activity". As a result, the prototype ended up with fewer activities than initially planned. Figure 5.11 illustrates the different activities which ended up being implemented in the application, and how they relate to each other.

What follows is a short description of each activity, its purpose and elaboration of some of the technical details. The complete source code for the application is found online at www.kaffemarius.com/OffPAD_Android_Application.tar.gz. If prompted for a username and password, *masterthesis* and *masterthesis2016* should be provided.

¹⁸Pencil - <http://pencil.evolus.vn/>

¹⁹MediaStore.ACTION_IMAGE_CAPTURE - https://developer.android.com/reference/android/provider/MediaStore.html#ACTION_IMAGE_CAPTURE



1. Main Activity invokes MediaStore.ACTION_CAPTURE_IMAGE
2. Upon success, MediaStore.ACTION_CAPTURE_IMAGE returns a reference to a captured image back to Main Activity
3. Main Activity invokes OCR Activity for OCR processing of the image previously returned from step 2. If the OCR activity during OCR processing is able to extract a valid transaction, the user is able to proceed to authenticate the transaction in question. If no valid transaction was identified, the user is informed, and requested to capture a new image.
4. OCR Activity invokes the Authenticate Activity which handles encryption/signing and communication to service
 - In case of successful authentication, user is informed in UI.
 - In case of unsuccessful authentication, user is informed in UI.

User is able to navigate between the different activities using on-screen buttons.

Figure 5.11: OffPAD Android application activities

5.2.1 Main Activity - MainActivity.java

The main activity is the starting point of the application, and is starting point of the application when it is launched. The main application offers a simple GUI where the user is able to start either the process of capturing an image (which will later be used for the OCR processing), or to start OCR processing (which requires a valid reference to an already captured image). The user interface provided by the main activity is seen in figure ??.

For the image acquirement, the built-in *MediaStore.ACTION_IMAGE_CAPTURE activity* (see previous section) is used, as seen in figure ?. The functionality for OCR processing has been implemented in the OCR Activity, which is



Figure 5.12: Main Activity UI - I

described in the next section. In addition to this, the user is able to customize some runtime settings, such as activate or deactivate automatic contrast increase for the images used in the OCR processing.

When the camera activity returns an acquired image to the main activity, the UI is updated to reflect this, as illustrated in figure 5.14.

5.2.2 OCR Activity - OCRActivity.java

The OCR activity is called from the main activity, using the “OCR” button in the UI. This is only possible when the main activity has received a reference to a captured image from an instance of the `MediaStore.ACTION_IMAGE_CAPTURE` activity, which comes with the Android SDK. The code snippet responsible for invoking the OCR activity can be seen in listing 5.3.

Listing 5.3: Main activity invoking OCR activity

```
public void dispatchOCRActivityIntent(View view) {
    Intent intent = new Intent(this, OCRActivity.class);
    if (mCurrentPhotoPath != null && !mCurrentPhotoPath.isEmpty()) {
        ...
        intent.putExtra("filePath", mCurrentPhotoPath);
        startActivity(intent);
    }
}
```

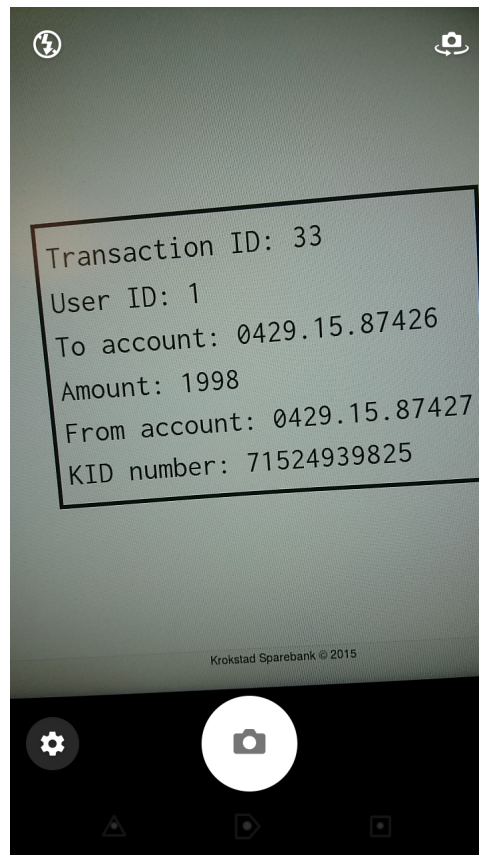


Figure 5.13: Built-in Camera Activity UI

```

else {
    Intent i = new Intent(this , Warning.class);
    i.putExtra("message", "No picture is active. Please capture an
        image before proceeding to OCR Analysis");
    startActivity(i);}
}

```

When the OCR activity is invoked with a reference to an image file, it will first set up the UI, before what is called an “asynchronous task” in the domain of Android programming is prepared for conducting the OCR processing. An asynchronous task in this context can be thought of as a simplified *thread* known from traditional computer programming. If we imagine a simple, single core CPU (Central Processing Unit), also just known as the “*processor*” in a computer, we know that it can only execute one instruction at the time. In this way, it would seem impossible that a computer would be able to do more than one thing at once. This is in fact true, yet while using a computer one can browse the web, listen to music and work on a text document *seemingly* all at once. What is actually happening behind the scenes is that each program is allocated short periods of computing time on the CPU which allows their instructions to be executed. It is usually the operating system which manages the process of this switching between programs, and it happens so often (at least for user experience centred operating systems such i.e., Windows or GNU/Linux)

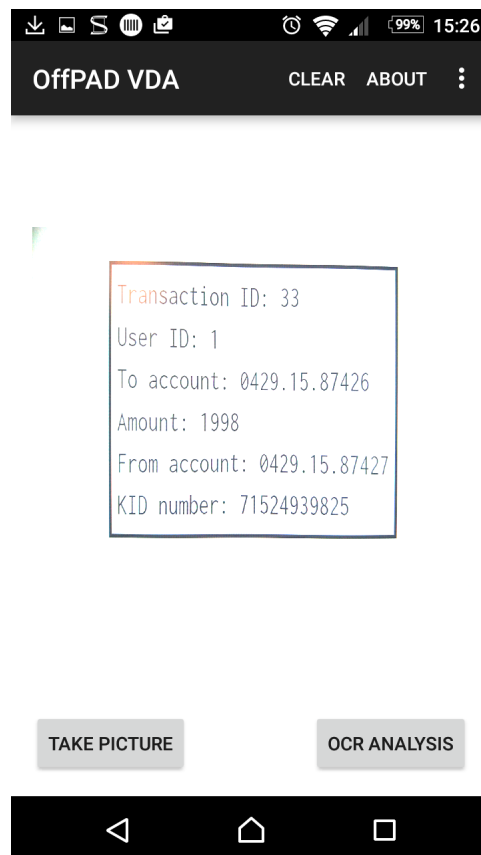


Figure 5.14: Main Activity UI - II

that it appears that multiple things are being executed in parallel.²⁰

The use of the asynchronous task allows the UI to be updated in order to give feedback to the user, while at the same time, OCR processing occurs. Some relevant parts of the asynchronous task can be seen in listing 5.4.

Listing 5.4: Asynchronous OCR task definition

```
private class OCRAnalysisTask extends AsyncTask<String, Void, Transaction>
{
    String TAG = "OCRAnalysisTask";
    ...
    Context context;

    public OCRAnalysisTask(Context context) {
        this.context = context;
    }

    @Override
    protected void onPreExecute() {
        progressDialog = new ProgressDialog(context);
        progressDialog.setMessage("OCR Analysis in progress. Please
            wait...");
        ...
        progressDialog.show();
    }
}
```

²⁰One can achieve true parallel execution using a single CPU with multiple cores, or multiple CPUs in a single computer.

```

    }

    @Override
    protected Transaction doInBackground(String... file) {
        String filePath = file[0];
        if (filePath == null) {
            return null;
        }

        TessBaseAPI tessBaseAPI = new TessBaseAPI();
        final String trainingDataPath = "/mnt/sdcard/no.kaffemarius.
            marius.offpadvda/";
        tessBaseAPI.setDebug(true);
        tessBaseAPI.init(trainingDataPath, "eng");

        ...

        Bitmap bitmap = ImageUtils.loadImageFromFile(filePath);
        bitmap = ImageUtils.increaseContrast(bitmap, 3, 0);

        if (bitmap == null) {
            return null;
        }

        tessBaseAPI.setImage(bitmap);
        String text = tessBaseAPI.getUTF8Text();

        ...

        tessBaseAPI.end();

        Transaction transaction = Transaction.parseRawOCRdata(text);
        return transaction;
    }

```

As seen in listing 5.4 there are references to two previously unmentioned classes, namely the *ImageUtils* (*ImageUtils.java*) and the *Transaction* (*Transaction.java*) classes. The *ImageUtils* class contains functions which were created to facilitate some pre-OCR image processing. As seen in listing 5.4, the function *ImageUtils.IncreaseContrast* is called before any OCR processing occurs, if enabled in the “Settings” menu.

This will increase the contrast of the image which, at least in my experience, will improve the accuracy of the OCR process. This code was originally written by StackOverflow user “Ruslan Yanchyshyn”²¹. There were also some ideas for other image manipulation functionality including, but not limited to automatic removal of visual noise²² and image binarization²³. During development, only the functionality to increase the contrast was implemented due to time constraints.

The second class referenced, the *Transaction* class is used to represent banking transactions within the application, and its definition can be seen in listing 5.5.

Listing 5.5: Transaction class definition

```

public class Transaction {

    private static String TAG = "Transaction";

    String transactionID;

```

²¹How to programmatically change contrast of a bitmap in android? - <http://stackoverflow.com/a/17887577>

²²Image Denoising - http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_photo/py_non_local_means/py_non_local_means.html

²³Binary Image - https://en.wikipedia.org/wiki/Binary_image


```

String userID;
String srcAccount;
String dstAccount;
String amount;
String KIDNumber;
String date;

...
}

```

As seen at the end of the asynchronous OCR task, a call to *Transaction.parseRawOCRdata()* is made, with the raw output received from the OCR processing sent as an argument. This function will attempt to parse and identify the different transaction fields from the text, while doing some validation of the data. If the data received cannot be a valid transaction (i.e., there are more fields in the data than what is expected), an error is returned. If the received data seemingly is a valid transaction, *Transaction.parseRawOCRdata()* will create a new instance of the Transaction class containing the transaction details extracted during the OCR processing, and return a reference to this object back to the caller.

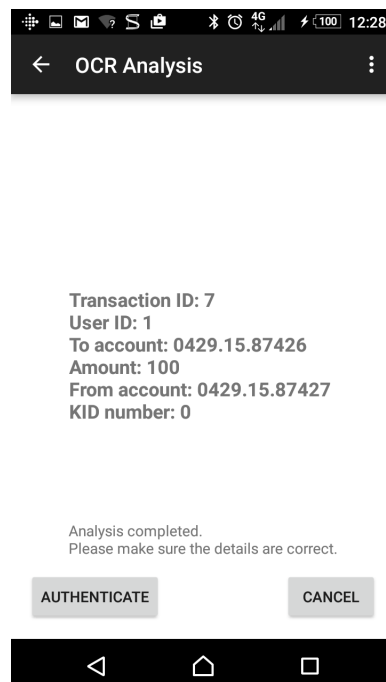


Figure 5.15: Screenshot - OCR Activity

When a valid transaction has been identified, it is displayed within the OCR activity's GUI, together with a friendly reminder for the user to verify the transaction details, as seen in figure 5.15. The user now has the possibility to proceed to authenticate the transaction towards the web server by invoking the *Authenticate activity*, by pressing the "Authenticate" button. The authenticate activity is described in the next section.

5.2.3 Authenticate Activity - AuthenticateActivity.java

The *Authenticate activity* can be invoked by the OCR activity when a valid transaction has been identified during OCR processing. This valid transaction, represented by an instance of the *Transaction* class, is made available to the authenticate activity upon invocation.

In this prototype, in contrast to the description provided by the OffPAD project, authentication begins immediately after the authentication activity has been invoked. According to the description provided by the OffPAD project, the user would have to provide some biometric identifier (such as a fingerprint) in order to authenticate the transaction. Due to the lack of a fingerprint reader on the first phone used for the development, this functionality was omitted.

As used in the OCR activity, an asynchronous task is used within the authentication activity for performing the authentication.

First, the preloaded keys, the server's public key and the user's private key, is loaded from the internal storage into memory. Then, a string representation of the transaction is created, using the *Transaction.toString()* function, seen in listing 5.6.

Listing 5.6: Custom toString function for the Transaction class

```
@Override
public String toString() {
    return transactionID + ";" +
        userID + ";" +
        date + ";" +
        srcAccount + ";" +
        dstAccount + ";" +
        KIDNumber + ";" +
        amount;
}
```

For example, calling the *Transaction.toString()* on the transaction object illustrated in figure 5.15, would produce a string similar to the one shown in listing 5.7. Note that the date field would vary, as it is calculated during execution.

Listing 5.7: String representation of transaction

```
7;1;1471950189;0429.15.87426;0429.15.87427;0;100
```

After a string representation of the transaction has been created, a digital signature²⁴ is created using the string representation together with the user's private key. Then, the string representation is encrypted using RSA encryption and the server's public key. PKCS#1 v 1.5 padding is used to remove the deterministic vulnerabilities found in textbook RSA as previously described in section 3.1.4 on page 40.

Then a single HTTP POST request is made to the web server, containing the user's user ID, the ID for the transaction the user wishes to authenticate, and the encrypted transaction and digital signature encoded using Base64 encoding²⁵, as seen in listing 5.8.

²⁴The digital signature produced is a SHA512 with RSA

²⁵Base64 encoding - <https://en.wikipedia.org/wiki/Base64>

Listing 5.8: HTTP POST request layout

```
data = "userid=" + URLEncoder.encode(transaction.getUserID(), "UTF-8") +
      "&transactionid=" + URLEncoder.encode(transaction.getTransactionID() , "UTF-8") +
      "&transaction=" + URLEncoder.encode(b64encodedTransaction , "UTF-8") +
      "&signature=" + URLEncoder.encode(b64encodedSignature , "UTF-8");
```

The web server will receive the HTTP POST request (unless any network error occurs), and attempt to validate the authentication, as described in section 5.1.2 on page 64. In the case of successful authentication, the UI will be updated and the user will be informed that the authentication has been validated by the web application, and that the transaction will be processed. In the case of an unsuccessful authentication attempt, the UI will update and the user will be informed that the web application found was not able to verify the authentication attempt based on the data received. The user is also informed to visit the bank's web site for more information. These two scenarios are illustrated in figure 5.16.

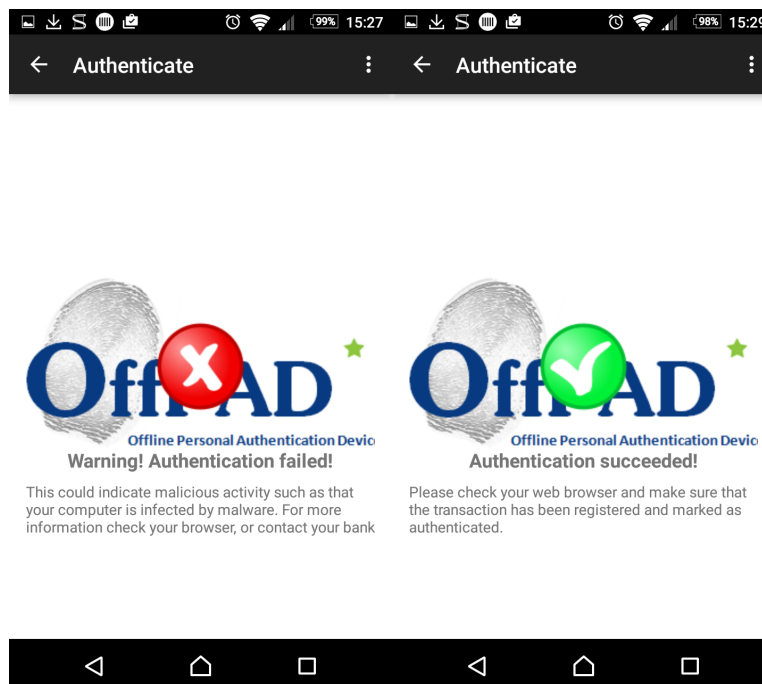


Figure 5.16: Authenticate Activity UI

5.3 Communication between webserver and smart-phone

Since the prototype web server is running as a virtual machine within a local computer, and the OffPAD application is running on a stand alone device (Android smart phone), it was required to have a way for these two separate components to be able to communicate in a lifelike way in order for the user experiments to be perceived as realistic for the participants.

By using *hostapd*²⁶ software, the virtual machine was configured to offer a WiFi network. This required a separate USB WiFi adapter²⁷ to be attached to the virtual machine, and act like network interface. The virtual machine was then configured to route any traffic received on this interface through the main virtual network interface using NAT²⁸ (Network Address Translation). Any traffic towards this interface, the main virtual network interface, would be routed through the physical network interface on the host machine, providing internet access to any devices connected to the wireless network provided by the USB WiFi adapter.

The *dnsmasq*²⁹ software was installed in the virtual machine, and configured to serve any DNS³⁰ (Domain Name System) request arriving to the interface hosting the wireless network. In this way, any attempts to resolve the domain name *krokstadsparebank.no* while connected to the wireless network would get the IP address of the virtual machine in response. This configuration is illustrated in figure 5.17 on page 83.

²⁶hostapd - <https://wireless.wiki.kernel.org/en/users/documentation/hostapd>

²⁷In our configuration, a D-Link DWL-G122 was used

²⁸NAT - <http://www.cisco.com/c/en/us/support/docs/ip/network-address-translation-nat/26704-nat-faq-00.html>

²⁹dnsmasq - <http://www.thekelleys.org.uk/dnsmasq/doc.html>

³⁰DNS - https://en.wikipedia.org/wiki/Domain_Name_System

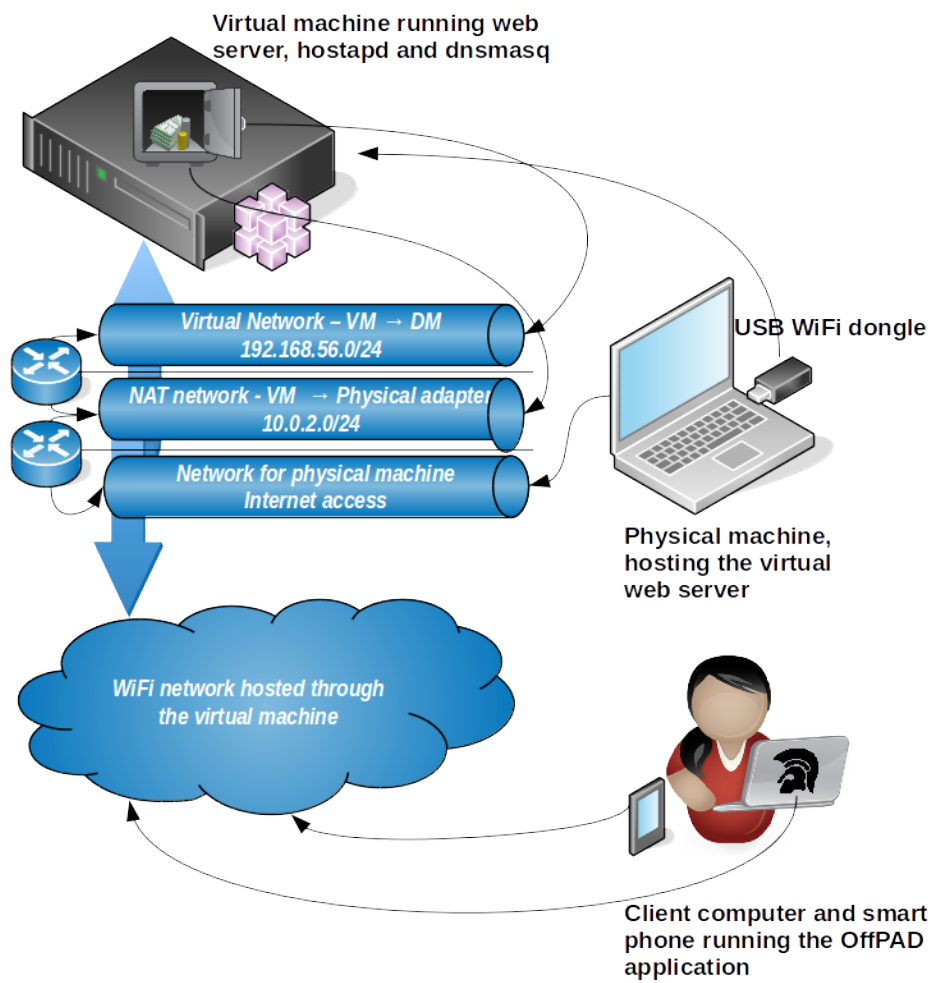


Figure 5.17: Setup configuration

Chapter 6

Testing

The focus for the usability study was to answer the third and final research question described in section 1.2 on page 7 in chapter 1: *“Which level of user friendliness can be achieved with this solution?”*, i.e., does the prototype provide a satisfactory user experience for users of the system. While working on this project, it soon became apparent that, due to time constraints, a full scale usability test would not be conducted. However, as with any HCI (Human-Computer Interaction) usability research, it is recommended to conduct what is called a *“pilot study”*.

A pilot study is a downsized version of the planned study, using a small number of participants. The objective of pilot studies is to reveal and identify any flaws in the design of study, in order for them to be corrected before a full scale study is conducted. [27, p. 292-294]. In addition, when working with a prototype, a pilot study may reveal crucial flaws in the design of the prototype itself.

This chapter is divided in four main sections. Section 6.1 addresses the way in which the study was designed. Section 6.2 describes how the study was conducted in practice. Section 6.3 addresses the key observations which were made during the study, as well as an attempt to analyse the data gathered in the study. The last and final section, section 6.4 addresses the question of how one can improve both the design of the study, as well as the design of the prototype, based on the observations made while the study was conducted.

6.1 Usability study – Design

The study consists of multiple participant sessions, where the subjects are asked to perform a set of tasks before answering a questionnaire. Before a participant session can begin, the subject is given an informed consent form (sometimes known as an institutional review board (IRB) form) which is to be read and signed, if and only if the subject approves the terms of the study. The motivation behind an IRB form, is to inform the subjects of the motivation of the study, what data is being gathered and how this data is being used. When signing an IRB form, the subject approves of the conditions described, and acknowledges that he or she are aware of

that they are participating in a research study, and consenting to it. The informed consent form can be seen in full in Appendix C on page 119.

For every participant session, a test environment is prepared. This environment consist of a desk with a computer display, a computer keyboard and a computer mouse. In addition, a smart phone preloaded with the OffPAD application is located to the left of the keyboard. The computer display is already displaying the user interface for the mockup e-banking system which has been developed as part of the prototype. The equipment is configured as previously described in section 5.3 on page 81. This setup is illustrated in figure 6.1.

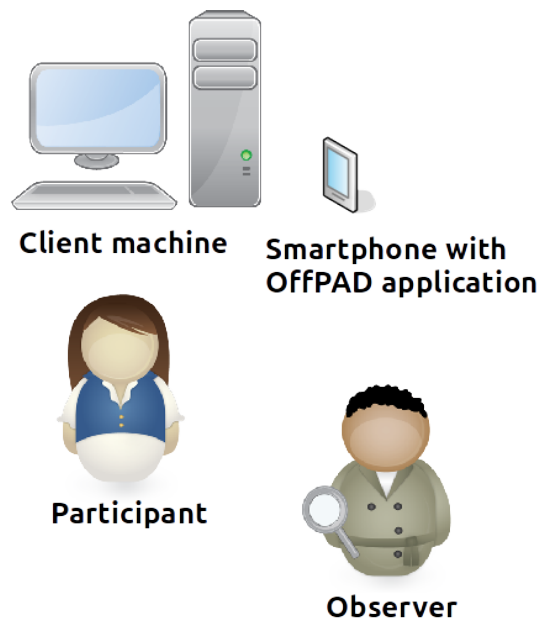


Figure 6.1: Participant session setup

If the subject, and only if, signs the informed consent form, the subject is placed in the test environment. Then the user is given a task list [17, p. 268-270], which can be seen in full in Appendix D on page 121. This task list instructs the user to register three different banking transactions, using the e-banking system provided in the test environment. One of these transactions is seen in table 6.1.

Amount:	6199,-
From account:	0429.15.87426
To account:	0429.15.87427
KID:	803113912605
Recipient:	"John Doe"

Table 6.1: Example of transaction defined in task list

The test environment is configured in a way so that the two first transactions will be registered without any issues, assuming that the OCR processing is able to extract the correct text from the image of the computer

display acquired by the subject. However, the KID number in the third transaction will trigger the web application to alter the transaction after it is displayed in the display for authentication, before it is inserted into the database. This is much like what would happen in a genuine Man-in-the-Browser (MitB) attack scenario. The motivation behind this, is to see if the subject, by following the cues provided in the system, would detect that something is wrong, and hopefully be able to identify that the transaction has been altered.

After completing the tasks defined in the task list, a questionnaire [27, p. 308] is presented to the subject, who is instructed to answer the defined questions. Questionnaires are a classic research technique from the domain of quantitative research methods. The use of questionnaires allows researches to collect data from a large number of participants in a relatively cost efficient way, and, given that the questions and options are correctly designed, the data gathered can be quantified and analysed using statistics. However, in order to produce any statistical valid results using questionnaires, there is a lower bound for the number of responses (i.e., how many subjects answering the questionnaire). According to Nielsen, the minimum number of participants when using questionnaires in order to get valid results is 20 [23].

The primary motivation behind the questionnaire, is to investigate whether the prototype is perceived as user friendly by the subjects. Since “*user friendly*” is a somewhat vague term, which often can be subjective and also relative to other existing, similar products, the questions are designed to investigate whether the OffPAD approach for authentication of e-banking transactions is perceived as *more* user friendly than existing solutions, such as BankID (please refer to 2.2.8 on page 28), or other similar technologies the subject may have had experience with. The questionnaire therefore consists of three main parts. Part one includes questions regarding what previous experience the subject has in regard of e-banking. All the questions in part one come with predefined answers, and the subjects are asked to circle one, or in some cases give multiple answers. Since we plan to compare the subject’s experience with the OffPAD approach for authentication of e-banking transactions against the subject’s experiences with other approaches, the first question is simply defined as “*Do you have any previous experience with e-banking?*”. Those of the subjects who answer “No” to this question are asked to continue to the questions in **part three** of the questionnaire. In this part the subjects are asked to rate agreement to three statements according to a predefined Likert Scale[27, p. 314]. As an example, question two in part III of the questionnaire is listed in table 6.2 below.

Since the plan is to compare the subject’s previous e-banking experiences to the experience with the OffPAD approach, only the responses where the subject has previous e-banking experience are relevant. In fact, the responses where the subject states not having any previous e-banking experience will be discarded, and not analysed. If the subjects are informed that if answering “No” to the question regarding previous e-banking experience, they are incapable of answering the rest of the questions and can

Question	Strongly agree	Agree	Neutral	Disagree	Strongly Disagree
I found it simple to register transactions using the e-banking system evaluated in the study					

Table 6.2: Statement from part III of the questionnaire

stop at the first question, then they might be intrigued to falsely answer “Yes”. This would introduce errors in the final dataset, which should be avoided. As a consequence, part III of the questionnaire was created, so that subjects without previous e-banking experience would have a set of alternative questions they can answer, although they will not be analysed or be part of the final results.

Those of the subjects who answer “Yes” in the first question, and have previous e-banking experience, are asked to continue with the questions in part I. The rest of the questions seek to identify what kind of technologies for data authentication they have experience with, such as BankID. Therefore, the subjects are asked in the questionnaire to rate agreement with statements such as *“Making bank transfers using my banks e-banking system is simple”* and *“I would prefer to use the OffPAD approach when registering bank transfers (transactions) over the the approach currently provided by my banks e-banking system”*. If a user finds a task easy to preform upon using a given system, one may conclude that the system is user friendly. If so, comparing the responses to these two statements in specific, could provide the answer to whether the OffPAD approach is perceived as user friendly by the subjects or not.

The subjects are asked to answer the questions in the questionnaire as honestly as possible. The questionnaire in full can be seen in Appendix E on page 123.

When a satisfactory number of participant sessions has been completed, the results from the questionnaires can be coded and analysed using a statistical approach.

6.2 Usability study – Pilot study

For the pilot study methods from both qualitative and quantitative research methodologies were utilized. From the qualitative domain the method of *observation* was used. During observation, users are observed by the investigator while performing a given task or a set of tasks. Observations can be used in any stage of product development, but when used early in the process, it can reveal critical flaws in both the design of the usability study (e.g., the questions in the questionnaires are too confusing), as well as in the design of the prototype itself (e.g., the web application contains a malfunction link) [27, p. 321]. As described in the previous section, the subjects are placed in the test environment and given a list of tasks which they are asked to perform. In the pilot study, the subjects were

observed while performing the tasks by an experiment observer. For the pilot study, it was decided to have five subjects. According to Nielsen, when conducting user experiments five is the ideal number of participants, from a cost-benefit perspective [24]. Nielsen also recommends to run as many tests as one can afford, however, I only conducted one.

Since the number of participants in a pilot study is far too low for the results to be statistically valid, the responses in the questionnaires would have little value. However, since the motivation for the pilot study was to identify potential flaws of the design of the study, this also includes the questionnaires themselves. In consequence, the questionnaires also have to be part of the pilot study. Any confusing or misleading questions could introduce bias in the results, so it is crucial that these questions are identified and corrected before any full scale study can be conducted.

In the pilot study, the subjects were asked to answer the questionnaires after completing the tasks defined in the task list. After filling out the questionnaire, they were encouraged to give the experiment observer feedback on the questions.

When a participant session was concluded, the questionnaire was collected and each was assigned a unique anonymous identifier, in the format of *P001* ... *P00n* before the process of cleaning and coding the data was started. The motivation behind these unique identifiers was that in case some typo or mistake was made during the coding of the data, it would be easy to identify where the mistake was made, and correct it.

The process of cleaning the data involved screening the responses for potential errors.

Before any statistical analysis could be conducted using the gathered data, it had to be coded in a way so that it could be interpreted by the software used for the analysis. In this case, it meant that each answer (or each combination of answers) had to be assigned a unique numeric value. In preparation for the coding process, a coding table for the questionnaire used in the pilot study was developed. The responses from the questionnaires were later coded according to this coding table, and plotted into a spreadsheet, as seen in figure 6.2. The coding table can be seen in Appendix F on page 127.

Subject	Age	Gender	1.1	1.2	1.3	1.4	2.1A	2.1B
P001	1	2	1	2	3	1	3	1
P002	2	1	1	6	6	3	3	2
P003	1	1	1	7	455	1	1	1
P004	2	1	1	15	35	3	2	4
P005	1	2	1	10	10	2	2	2

Figure 6.2: Excerpt of coded data

6.3 Pilot study – Results

6.3.1 Pilot study – Observations

All subjects were able to, just by looking at the web UI, figure out how to register new transactions. However, some of the subjects had some issues registering some of the transactions in the web UI because it was unclear how exact they were to copy the transaction details from the task list. For example, one subject provided the account number without any periods (i.e., 04291587426 instead of 0429.15.87426). This caused the web application to reject the transaction registration, since it was unable to find a matching account. The web application should be able to handle this scenario, presumably by adding some extended validation and interpretation of the data provided by the user. Since the number of digits in account numbers are fixed, the web application could automatically attempt to add the appropriate periods automatically, or simply ignore all periods.

Some subjects experienced a similar issue when providing information in the amount field. Some provided the literal string 200,-, instead of just the digits 200. This also caused the web application to reject the transaction. Additional logic should be implemented in the web application to handle these scenarios.

A bug causing the web application to raise a MySQL error was also discovered during the first participant sessions. As instructed in the task list, the subject was to leave the KID field empty when registering the second transaction. Due to a last minute bug fix concerning another issue, the web application did not handle an empty KID field. Since this was indeed a major flaw in the prototype itself, subsequent participants were informed of the issue.

For some of the subjects, it was not intuitive when the OffPAD Android application was to be used. This indicates that there should be some kind of visual cue in the web UI, at least the first time a user attempts to register a transaction, that the user will have to utilize the OffPad Android application for authentication.

Also, when acquiring images of the registered transaction displayed in the browser window, several of the subjects failed multiple times to get an image of sufficient quality. Some images were taken too far away from the display, making the text too small for the OCR processing to succeed. In other cases the picture was taken with either a horizontal or vertical tilt, which had been shown from our previous OCR experiments (please refer to section 3.2 on page 42) would affect the performance of the OCR processing. In some of the cases, the mouse pointer was located on top of the transaction details at the time when the picture was taken, which would cause the OCR processing to extract incorrect text from the image. In this way, it came apparent that the instructions provided to the user regarding image acquisition are insufficient, and should be improved. Also, some of the image optimisation features proposed for the OffPAD Android application (please refer to section 5.2.2 on page 75) could improve

this aspect of the prototype a great deal when implemented.

Some of the subjects experienced some instabilities in the OffPAD Android application itself when attempting to proceed to OCR analysis after an image had been acquired. The application would in some cases simply crash, informing the user that “*Unfortunately, OffPAD VDA has stopped*”. This indicates that an *unhandled exception* was raised during runtime. In Android (or Java programming) for that matter, an exception is an event which occurs during runtime, which disrupts the normal execution of a program. Exceptions can occur for several reasons for example I/O errors, invalid user input or invalid format for some piece of information being processed by the program itself. In some cases exceptions are expected, and developers can deal with them by writing special blocks of code which essentially tells the program how to behave when an exception is raised. For example, if a program is designed to read some information from a specified file into memory, but the file does not exist in the specified location, a **FileNotFoundException**¹ can be raised. In this scenario, developers could add a code block instructing the program to inform the user, and ask him or her to specify a new file path where the file can be found. Unhandled exceptions are simply exceptions which occurs, where the program does not have a pre-programmed way of dealing with the exception. This usually causes the execution to come to a halt. Unfortunately, the smartphone device itself was not connected to a computer during the participant sessions, so no debug information regarding these crashes were gathered. At the time of writing, what caused these crashes is still unknown.

When a transaction had successfully been authenticated using the OffPAD Android application, the impression after the observations is that the visual feedback given within the OffPAD Android application is sufficient. That being said, the majority of the subjects seemed to expect that the web UI would automatically update whenever a transaction was successfully authenticated, which it did not. When the display in the OffPAD Android application showed the confirmation screen, the subjects would turn the web UI, for a similar confirmation there. This is something which should be implemented in the prototype at a later point.

For the transaction where authentication failed, either due to incorrect OCR extraction that the subjects failed to identify, or due to the planned in-browser alteration of the transaction details, the visual feedback given by both the browser and the OffPAD Android application seems to be insufficient. The subjects understood that *something* was wrong, but were unable to interpret the situation correctly using the information provided by the system. The warning found in the OffPAD application’s display seem too small, because none of the five subjects read the warning when a failed authentication attempt occurred. Since the web UI did not update or provide any additional feedback, the subjects seemed rather confused whenever a failed authentication attempt occurred. As already stated, the

¹FileNotFoundException - [FileNotFoundException.html](http://docs.oracle.com/javase/7/docs/api/java/io/FileNotFoundException.html)

web UI should implement additional feedback, which would help in this scenario as well. However, it is clear that the visual feedback given in the OffPAD Android application has to be improved as well. One possible solution is to display the warning in a more distinct way, i.e., using a separate activity to display the warning.

While answering the questionnaires, the participants had the opportunity to consult the experiment observer if they found any of the questions confusing and/or misleading. As a consequence, some flaws with the questions were identified during the pilot study. Question 1.2 (*“What means for user authentication is available in the e-banking system most frequently used by yourself (i.e., how do you log in to the e-banking system)”*) and 1.3 (*“What means for data authentication is available in the e-banking system most frequently used by yourself (i.e., how do you authenticate banking transactions)”*) led to some confusion. First and foremost, it did not seem like the subjects were able to tell the difference between user and data authentication without further context. This led some of the subjects to circle the incorrect selections. Further, some of the subjects were unaware of the correct terminology used for the different technologies available in their banks, which led to some additional confusion. In order to minimize the risk for misunderstanding and misinterpretation regarding these questions, one could perhaps add images of the different technologies (e.g., OTP generator, mTAN and BankID on mobile) to the corresponding selections. This could make it easier for the subject to relate to the different options, and lead to less confusion.

Also, when asked to rate agreement with the sentences in 2.1, such as *“I would consider to use the OffPAD approach when registering bank transfers (transactions), if available in my bank e-banking system”* and *“I would prefer to use the OffPAD approach when registering bank transfers (transactions) over the approach currently provided by my banks e-banking system”*, some critical feedback was received. The majority of the subjects asked the experiment observer something along the lines of *“When answering this, should I base my answer on the current state of this prototype – or the general concept? Because I like the concept.”*. This indicates that although the subjects were positive to the approach to authentication of e-banking transaction showcased in the prototype, the overall quality of the prototype itself is not satisfactory, and did not provide a good user experience.

6.3.2 Pilot study – Analysis

As stated numerous times, the number of participants in the pilot study is *too low for any statistically valid results*. However, since the pilot study had already been conducted, I decided to try to analyse at least some of the gathered data regardless. At least, this would showcase how the responses would be analysed if a study is to be conducted using a sufficient number of participants.

In the the questionnaire, all subjects acknowledged previous e-banking experience (question 1.1). As previously described, the subjects had some confusion regarding question 1.2 and 1.3, which is assumed to cause incorrect responses. As a consequence, further analysis of the responses

Both (BankID and BankID on mobile)

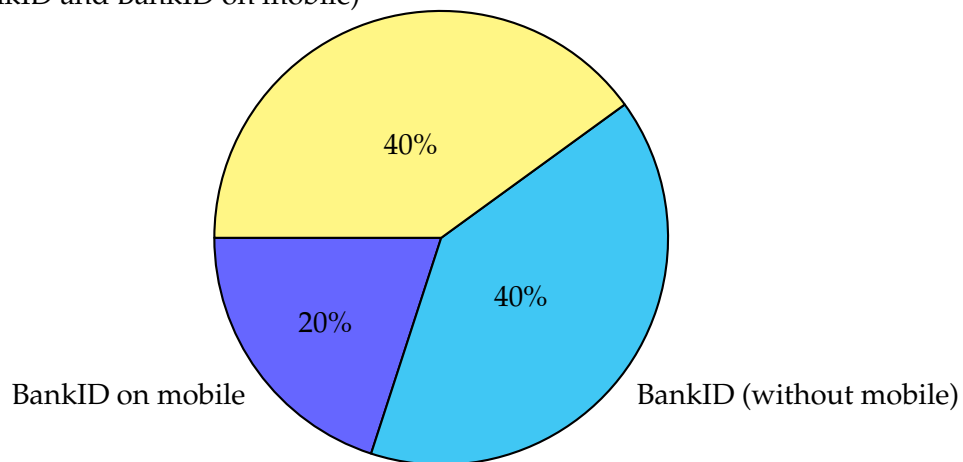


Figure 6.3: Results from questionnaires - Question 1.4

for these questions are omitted. These questions, as previously described, should be modified so that they appear more clear for the subjects.

According to the responses for question 1.4, all subjects had previous experience with Bank ID. The following chart illustrates the distribution in previous BankID experience among the subjects.

The results from the second part of the questionnaire is seen in figure 6.4 on page 94. The responses have been summarised, before a average was calculated. As seen in the results, the participants had a high level of agreement with statements such as *"Making bank transfers using my banks e-banking system is simple"* (Question 2.1B) and *"The approach used to authenticate banking transfers (transactions) in my banks e-banking system is user friendly"* (Question 2.1C), resulting in a average of respectively 2 ("Agree") and 2 ("Agree"). However, when rating agreement with statements such as *"I would consider to use the OffPAD approach when registering bank transfers (transactions), if available in my bank e-banking system"* and *"I would prefer to use the OffPAD approach when registering bank transfers (transactions) over the approach currently provided by my banks e-banking system"*, the result yielded an average of respectively 3.2 and 3.2. According to the pre-defined Likert scale, this corresponds to a response between "Neutral" and "Disagree". These results support what was observed during the participant sessions – the subjects do not find the approach provided by the prototype for authentication of e-banking transactions sufficiently user friendly.

6.4 Usability study – Suggested improvements

Based on the observation made during the pilot study, I suggest the following improvements to the prototype itself, as well as to the design of the usability study itself:

- Some bugs in the prototype were found throughout the participant

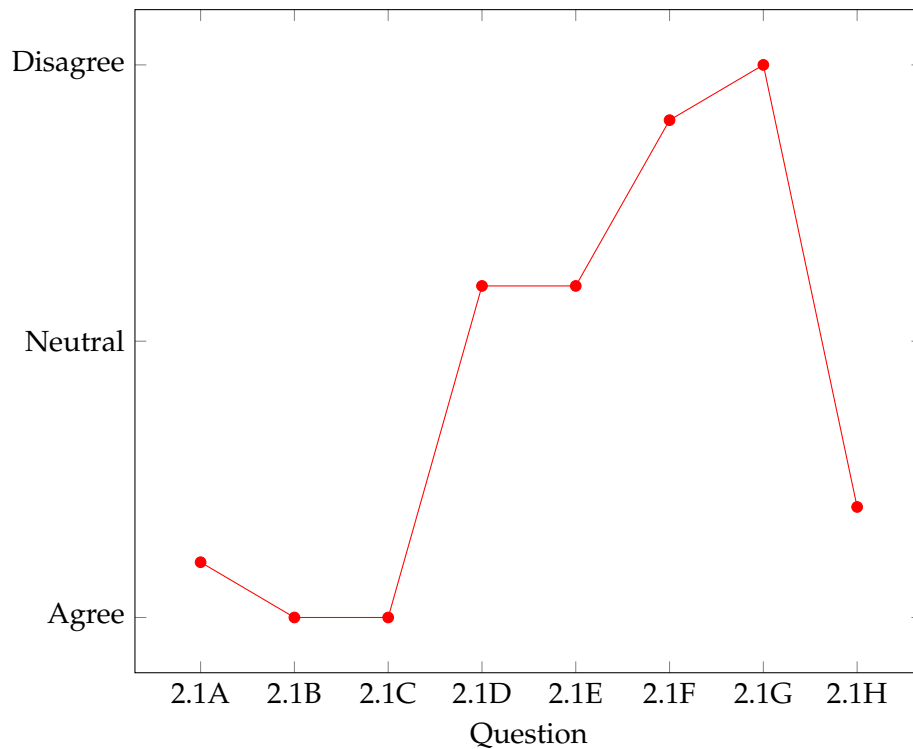


Figure 6.4: Results from questionnaires - Part II

sessions. For some of which, the root cause is identified and should be fixed. For others, the root cause is still unknown, and should be further investigated.

- Additional feedback in the web UI. As revealed in the pilot study, the users expected some sort of visual cues in the web browser when a transaction is successfully authenticated, or an unsuccessful authentication attempt has been made.
- Additional instructions regarding how to acquire an image of the transaction details displayed on the computer display. These instructions could be found in the web UI, in the OffPAD Android Client or both.
- The image optimisations features described section 5.2.2 should be implemented and tested. We assume that these optimisation will improve the source image data used in the OCR processing, thus increase the performance and accuracy of the text extraction while decreasing the number of attempts to acquire a source image of sufficient quality. It is assumed that this would provide an improved user experience.
- As previously described, some of the subjects found some of the questions found in the questionnaire confusing, which could lead to

errors in the dataset. These questions should be formulated in a way so that the risk for confusion and misinterpretation is minimised.

Part III

Conclusions

Chapter 7

Conclusion & future work

7.1 Goal fulfilment

In the first part of this Master's thesis, a set of research questions was defined (please refer to section 1.2 on page 7). I have done my best to answer these research questions by the analysis and investigations described in the previous chapters. In order to make it more explicit, this section will summarize the results from this Master's project, and how they are related to the previously defined research questions.

- **Q1:** *How can, based on concepts and ideas derived from the OffPAD project, a solution for data authentication of e-banking transactions be designed?*
- **A1:** Based on high-level concepts and ideas from the OffPAD project and its members, combined with own ideas developed while working on this Master's project, a practical design for a system for data authentication has been described in in chapter 4 on page 53.
- **Q2:** *Based on this design, how can it practically be implemented in a working prototype?*
- **A2:** As described in chapter 5 on page 61, a working high-fidelity prototype has been implemented. This prototype seek to showcase how the OffPAD approach to data authentication using OCR, rather than the OffPAD device itself. The prototype consists of a smartphone application developed for the Android platform and a mock-up e-banking web application.
- **Q3:** *Which level of user friendliness can be achieved with this solution?*
- **A3:** In order to evaluate the level of user friendliness the solution provides, a pilot usability test study was conducted. Based on the results from the pilot study, in its current state, the solution for data authentication described in this document does not provide a satisfactory level of user friendliness. The subjects who participated in the pilot study seem to like the concept itself, but the prototype developed through this project was not of sufficient professional quality.

7.2 Future Work

As when working on any major project, new ideas and perspectives are developed throughout the process. Despite the fact that the prototype which has been developed during this Master's project at the time of writing is not of a sufficient professional quality, I feel that the approach to data authentication for banking transactions in e-banking it provides, is promising.

In addition to e-banking, I feel that this approach has great potential, and can be applied to other domains as well. These domains include, but are not limited to e-voting, e-health and e-government. Since the approach for data authentication described throughout this document in theory could work in any use case where visual text is involved, it may be applicable to more analogue use cases as well.

As previously described, several areas of improvement regarding the prototype itself have been identified while working on this project. It is assumed that the pre-OCR image optimisation features proposed for the OffPAD Android application would improve the user experience for any users of the this system. Further, a general improvement in the stability of the OffPAD Android application and more appropriate feedback in the web UI could make both users and service providers more receptive to the technology. It would be interesting to see a more thorough usability study of the prototype when all the described improvements are implemented, and what the results of such a study would be.

Throughout this thesis, it has been assumed that the OffPAD has already been initialized by the service provider, and is ready for use when received by the user. This thesis does not discuss important questions such as how this initialization should occur, and how keys and biometric modalities are being handled. My personal opinion is that in order for the OffPAD to be adopted in a broader sense by both the industry and users, a common secure model for initialization and key management has to be developed.

Further, attempts to break the data authentication scheme described in this text should be conducted. Since security is a continuous process, rather than a constant state, vulnerabilities in this scheme could exist – they might not have been identified *yet*.

Appendix A

Deliverable 5.2A

OffPAD: Deliverable D5.2.a

– Description and Design of the software modules for the authentication classes Data-US

UiO OffPAD team

1 Introduction

This deliverable presents a detailed description of the design for the authentication class Data-US and the interface of the respective software module that implements it. We first recall from the previous Deliverable 5.1 what the class Data-US is supposed to do, in general terms. We then proceed to describe the details of this security ceremony. We use in our description the formalism of Actor Network Procedures, which we studied earlier in this project (see the paper [10]) and has been previously included as part of the Deliverable 5.1. We also make use of Sequence Diagrams when we make descriptions more close to the software implementation level. We also present the API that this module provides or requires from the rest of the OffPAD architecture (i.e., from the OffPAD hardware device, software agent App, or platform).

The use of Actor Network Procedures has the benefit of being graphical, besides having a formal language presentation. Moreover, the graphical notation is close to what security protocols designers are used with and at the same time close to sequence diagrams, as can be seen further on. The formal language associated would allow us to do formal analyses of the security ceremony that we define. But this analysis is not included in this document, and will make the subject of a future deliverable.

We focus here on the ceremony which is described as:

Data-US: Authentication of user data by the service provider, to be based on OCR (Optical Character Recognition) done on the OffPAD secure hardware.

2 Data Authentication

2.1 Background and motivation

According to the X.800 standard the concept of data origin authentication is *"the corroboration that the source of data received is as claimed"* [4]. This is different from entity authentication because knowing the identity of a remote entity in a session (entity authentication) is different from knowing whether the data received through a session with a specific remote entity genuinely originates from that entity. This difference might sound subtle at first glance but it is in fact fundamental for security, as explained below.

Malware infection of client platforms opens up for attacks against data authentication that entity authentication can not prevent. More specifically, entity authentication is insufficient for ensuring trusted interaction in case the client platform is compromised. A typical example is the situation of

online banking transactions with mutual entity authentication. Even when there is strong 2-factor user authentication, and we assume that users correctly interpret server certificates for server authentication, there is the possibility that malware on the client platform can modify data communicated between client and server platforms, and thereby compromise transactions. Current attacks against online banking are typically mounted in this way. Such attacks lead to breach of data integrity, but not a breach of entity authentication.

The preparation for this type of attacks typically includes tricking the user into installing a Trojan, i.e. a program that really or seemingly does something useful, but that in addition contains hidden malicious functionality that allows the attacker to take control of the client platform. During an online banking transaction the attacker uses the Trojan program to change transaction details without the user's knowledge. SpyEye, Zeus, IceIX, TDL, Hiloti, Carberp, and many others [3, 8] are concrete examples of malware that enable such attacks.

The separation between the human/legal entity and the system entity on each side of a client-server session – as we described earlier in Deliverable 5.1 – makes it necessary to specify which entity in particular is the assumed origin of data. In case e.g. the human user is assumed to be the origin, and the client system modifies data input by the user before it is sent to the server system, then this would be a breach of data origin authentication. However, in case the client system is assumed to be the origin, the same scenario would not be a breach of data authentication. The general rule is that the object of entity authentication must also be the origin of data authentication. For typical online transactions where the human user is directly involved and authenticated, the user must also be seen as the origin of data for data authentication purposes. Unfortunately current solutions for user data origin authentication are either non-existent or inadequate because they assume the client system to be the origin of data [2].

Users rely on visual cues to know whether a browser session is secured with TLS. After verifying that TLS is enabled, averagely security aware users will comfortably input their banking account and transaction details into the browser window. However, many users ignore that malware like those mentioned above has functionality commonly known as a "*web inject*" that can change the behaviour of the browser and modify input and output data arbitrarily.

The fact that entity authentication and data authentication are two separate security functions implies that it is necessary to have specific security mechanisms to ensure data integrity in online transactions. The OffPAD enables data origin authentication with high assurance and usability, as explained below.

2.2 High-level description for Data Authentication Supported by the OffPAD

Users generally rely on what they see on a computer display to read the output of transactions, to verify that they type correctly, and to ensure that the data being sent through online transactions is according to their intentions. In general, all this depends on the integrity of the computing platform to which the VDU (Visual Display Unit) is connected. Assuming that the computing platform is infected with malware it is *a priori* impossible to trust what is being displayed to be 100% correct [1, 6, 7, 11, 12].

The prospect that the computer display can lie to us is both frightening and real. This problem is amplified by the fact that we often read data from platforms that are not under our control, and

that hackers have incentives for trying to manipulate the systems and the way data is displayed. For example, typical attacks against online banking consist of using a malicious Trojan on a client computer to send falsified transaction data to the bank server while displaying what the user expects to see.

In order to provide data authentication, some online banks offer SMS authorization of transactions, which consists of mirroring the received transaction data (destination account and amount to be transferred) together with an authorization code by SMS to the user. After verifying the correctness of the transaction data, the user returns the authorization code via the browser to confirm that the integrity of the transmitted transaction data. In case of an attack, it is assumed that the user will notice when transaction data have been changed, so the attack can be stopped by canceling the transaction. This method can in theory provide strong data authentication, but it puts a relatively high cognitive load on the user for verifying the transaction data. In a study, it has been shown that about 30% of users fail to notice when transaction data in an SMS message have been changed, which means that 30% of attacks would succeed even in the presence of SMS authorisation [2]. The problem with SMS-authorisation is poor security usability, which fortunately can be solved with Lucidman as is explained below.

Fig.1 illustrates a simple ceremony for data origin authentication, i.e. to ensure that what is displayed on the VDU corresponds to what is being transmitted to other parties in online transactions. The method assumes that the user has an OffPAD with integrated camera, OCR (Optical Character Recognition) and communication functions. The user first captures a screenshot from the VDU with the OffPAD camera, then uses the OffPAD to recover the displayed data from the image through OCR, and finally to compute a MAC (Message Authentication Code) which is sent along with the original transaction data. The MAC enables the recipient server to authenticate the received original data.

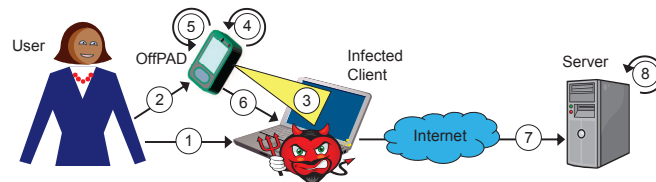


Fig. 1. Ceremony for data authentication with the OffPAD

The actions/messages of the ceremony are described in Table 1.

Even though it is assumed that the client platform is infected, it is easy to see that attackers will not be able to falsify the transaction data undetected. Falsified transaction data would produce a MAC mismatch, which would be discovered by the server in (8).

In order to successfully falsify data, the attacker would have to compromise both the client platform and the OffPAD simultaneously. Since the OffPAD is offline, it is assumed that the OffPAD will not be exposed to threats from the Internet. A more detailed threat analysis is provided in the next section.

3 Detailed Description of the Data Authentication Ceremony

We will present first the structure of the ceremony and discuss its components (also relating to the other documents provided by the partners, when needed). We then continue to describe the secure

Nr.	Message / action description
1.	User types the transaction data in a browser window on the client computer
2.	User activates the OffPAD to take a snapshot of the browser window
3.	Snapshot is taken of the text displayed in the browser window on the VDU
4.	The OCR function recovers the transaction data from the snapshot
5.	MAC generation with the transaction data and the user-password as input
6.	OffPAD send the MAC to the client computer
7.	Client computer sends transaction data together with MAC to server
8.	Server verifies that the MAC corresponds to the received transaction data.

Table 1. Sequence of messages and actions for data authentication ceremony

run that we see of this ceremony. Any other possible runs should be considered insecure. In the end we discuss the API associated to this ceremony and present some sequence diagrams. These would be similar, but more concrete, to the description of the secure run.

3.1 Structure of the Security Ceremony for Data Authentication

The structure of the Data ceremony is pictured in Figure 2. It comprises of:

Identities of the actors involved in the ceremony. Here we have two: the **User** (U) and the **Bank** (B). For visual aid, we encircle all the configuration under the control of the same agent.

Configurations which are the various parts involved in the ceremony, be that devices or human persons. Here we have:

- Five *basic configurations*, which are depicted as the black circles, and labelled according to their purpose. From right to left, these are:

S_B which represents the Server of the Bank. This is why this configuration is under the control of the Bank identity (represented as the under-script). In Figure 1 this is represented as the server box.

Term which represents the Terminal to which the user is supposed to interact in order to perform the transaction operation. In Figure 1 this is represented as the laptop which may be infected with malware. But the terminal can be other things as well, e.g., the Internet Browser that is used to communicate with the bank, or the SmartPhone, or an App on the SmartPhone.

Note that the Terminal is not under the control of any of the two identities. Therefore it is left open to potentially become under the control of an Attacker.

At this level of description it is good to leave room for such variations. But at the later point when we describe the sequence diagram we may become even more concrete.

OA_U which represents the OffPAD Software Agent that is supposed to be deployed on the Terminal (or otherwise) to facilitate the interaction of the Terminal with the OffPAD Trusted Hardware **OH_U**. The software agent is assumed to be under the control of the User.

P_U which represents the Persona of the user (i.e., the human part of the ceremony). We do not focus on the concept of Personas here, but consider it as is usually done to represent

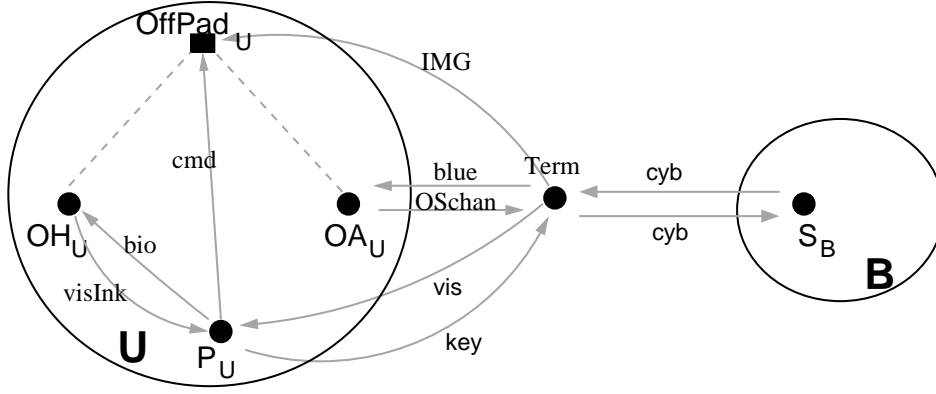


Fig. 2. ANP **structure** for the Data authentication ceremony.

the Human User involved in the protocol. For more details about Personas please see our paper [5] and the references therein.

OH_U which represents the OffPAD Trusted Hardware which is assumed to offer the following trusted hardware components: computation power, biometric input in the form of Fingerprint Reader, visual output in the form of InkDisplay, and NFC communication capabilities.

- One *complex configuration*, (depicted as a small black square) which we will call the **OffPAD_U** being formed of putting together the OffPAD Hardware and the OffPAD Agent to share information between each other. The dashed lines symbolize the complex configuration and how information can flow from the components to the back square and back. This complex configuration is thus under the control of the User. How the sharing of information is done is not so important as long as it respects the assumption that it is secure, i.e., it cannot be intercepted by an attacker. For this the TazTag will implement this sharing using NFC communication.

The precise implementation of the sharing of information between the Hardware and the Agent is important, and will be detailed further.

Channels which represent the various ways of transmitting information or interacting in some sort between the parts of the ceremony (i.e., between the configurations). The channels are depicted as grey arrows, pointing in the direction of the flow of informations, and are labelled with the type of channel. The type of channel encapsulates the security assumptions we have about this interaction medium, and we will make these precise. We are working on developing a formalism for this, which can be used in conjunction with the formalism we have presented in [10] for the rest of the AMP ceremony to achieve formal verification.

The ceremony comprises of the following channels:

- There are two **cyber-channels** (cyb) between the Server of the Bank and the Terminal. Cyber channels carry the assumptions that are un-safe, falling under the standard Dolev-Yao model of attacker which can listen to messages, change messages, and insert new messages.
- There is a **visual** channel (vis) between the Terminal and the Persona. This can be achieved through any standard display screen, as that of a laptop or of a SmartPhone. The assumption is that nothing can change the messages sent on this channel, and thus the Persona receives whatever is sent by the Terminal. But this does not exclude that an infected Terminal sends spurious information on this channel.

- There is also a **keyboard** channel (key) which can be used by the Persona to input information to the Terminal. Similar assumptions as with the visual channel hold here as well.
Note that in both the above channels we do not consider any other aspects like the **drivers** which handle the information processing for these channels on the Terminal side. We consider these drivers to be part of the Terminal. Therefore, if the Terminal is under the control of the Attacker these drivers may be as well, and thus handle information in unexpected ways. For example, we may type one character at the keyboard, but the driver changes it into another; this may not be observable either because the visual display driver is manipulated as well, or just because of the situation, like with a password form field where instead of seeing what we type we just see star symbols.
- There is also a **visual InkDisplay** channel (visInk) from the OffPAD hardware to the Persona. Assumptions are made of this channel similar to the other visual channel. The InkDisplay is though more rudimentary and thus can display more simplistic information, and maybe without all the visual aids of a fully-fledge visual laptop display. For example colours are not available here, so any visual aids that are usually used to help users, like color red for danger/unsecure, or color green for secure. This also may make the communication of the information to the user more error prone, in the sense that the user may not distinguish well the information displayed. Such situations can be taken advantage by an Attacker in a Social-engineering attack.
On the other hand, the fact that this channel is between the OffPAD hardware means we can add more assumptions and say that the drivers that handle the information displayed cannot be manipulated.
- A **biometrics** channel (bio) transfers Fingerprint biometrics from the Persona to the OffPAD hardware.
- We have added also a **commands** channel (cmd), which communicates very simplistic command messages, OK, StartTransaction, StartOCR, etc. Such a channel can be devised in any number of ways, like through buttons, or through a led-blink and finger press protocol. In any case the simplicity of the messages makes it difficult to attack such a channel. Even more in our situation where this channel is between the Persona and the OffPAD configuration, which is assumed to be secure and under the control of the User.
- There are two channels between the Terminal and the OffPAD Agent to handle any kind of information transfer. From the point of view of the information these channels are like cyber-channel, being able to transfer any kind of data. But these channels are implemented with short-range communication technologies like **NFC, Bluetooth, or Operating System communication channels**. NFC and Bluetooth would be used when the Agent and Terminal are part of two different devices (i.e., the Terminal is a Laptop), whereas OS-channel would be used when the Agent is an App on the SmartPhone. This last situation is what we concentrate on in our first prototype.
- There is also a **IMG** channel for transferring an image equivalent of the visual display (or part of it) of the Terminal to the OffPAD. How this channel is achieved can vary, and we are still investigating which method is best for our purposes. The assumption is that the image transferred is identical to the one displayed on the visual display (i.e., send on the visual channel to the Persona). One example, in the case of Terminal being a Laptop, could be to use a camera incorporated in the OffPAD hardware (which is trusted) and to take a picture of the Terminal screen. Another example, in the case the Terminal is a SmartPhone, can be to take a screen-shot and guarantee that the OS-transmission path for this operation to the OffPAD Agent is secured. This means that all involved software components from the Operating System

of the SmartPhone (like drivers or screen-capture software) are ensured to not be susceptible to Attacks which have the power to alter the information. Note that attacks that only have the power to snoop this channel do not break our security assumption.

3.2 Secure Run of the Ceremony for Data Authentication

After we know the structure of the possible communications between the components of our ceremony we can detail the secure run that we expect of this security ceremony. Any other runs, if possible, are considered insecure. We picture this run in Figure 3 with graphical notation introduced in [9] and which we investigate in [10]. This notation should be familiar from Strand Spaces. But otherwise, the notation is rather intuitive and also similar to sequence diagrams. Nevertheless, we explain each step of the run, because for our ceremony there are various assumptions and discussions which do not appear in the figure.

In a run we draw horizontal arrows when there is communication between the parts of the ceremony. These parts are the ones defined in the structure Figure 2 and for the run we just display these parts at the top. One can imagine a vertical line (as in sequence diagrams) to pertain to that specific actor in the ceremony. We label the horizontal arrows with the channel name on which the communication is done (shows in the same grey color as the arrow itself). On top of the arrows we display the information that is being sent, or the action that is being taken or that produces the respective information.

We display internal computations or actions by a vertical arrow and label it with the respective activity that is being performed (possibly with the outcome of that activity).

Sharing of information inside the complex configuration is represented by the same dotted line. In our case the sharing between the Software Agent and the Hardware is done through an NFC channel

We detail more each step in the above run.

- The ceremony starts with the service provider sending a fresh value (i.e., the down-arrow labelled by $nu[x]$, where x is the value) through the Terminal to the OffPAD configuration which shares it with its components. This is to be used as unique identifier of this particular Transaction.
- The Persona inputs through the keyboard channel to the Terminal the Transaction information, which we denote by the general term t . What exact structure this information has is not relevant here, and can later be defined to be many things, like bank account and amount to be transferred.
- The Terminal displays this information back to the User through the visual display channel. This is natural because usually the user fills in a form, which is already displayed. But we cannot assume that the Terminal displays the same information, therefore we denote this by t_1 .
- We then ask the Persona to verify that the information that she intended (and typed) is the same as the information that she sees. This part is actually done involuntarily (or not done at all by a careless Persona). But it is good to have this step part of the ceremony, though it does not appear in the implementation part that we see later.
- Presumably, the Persona then starts the OCR operation by sending such a command to the OffPAD.
- In turn, the OffPAD Software Agent is the one asking the Terminal for the image related to the transaction form.
- The acquiring of the Image is an operation that can be done in various ways, and we let it open here. But in any case we can assume that the image is being transferred from the Terminal (where

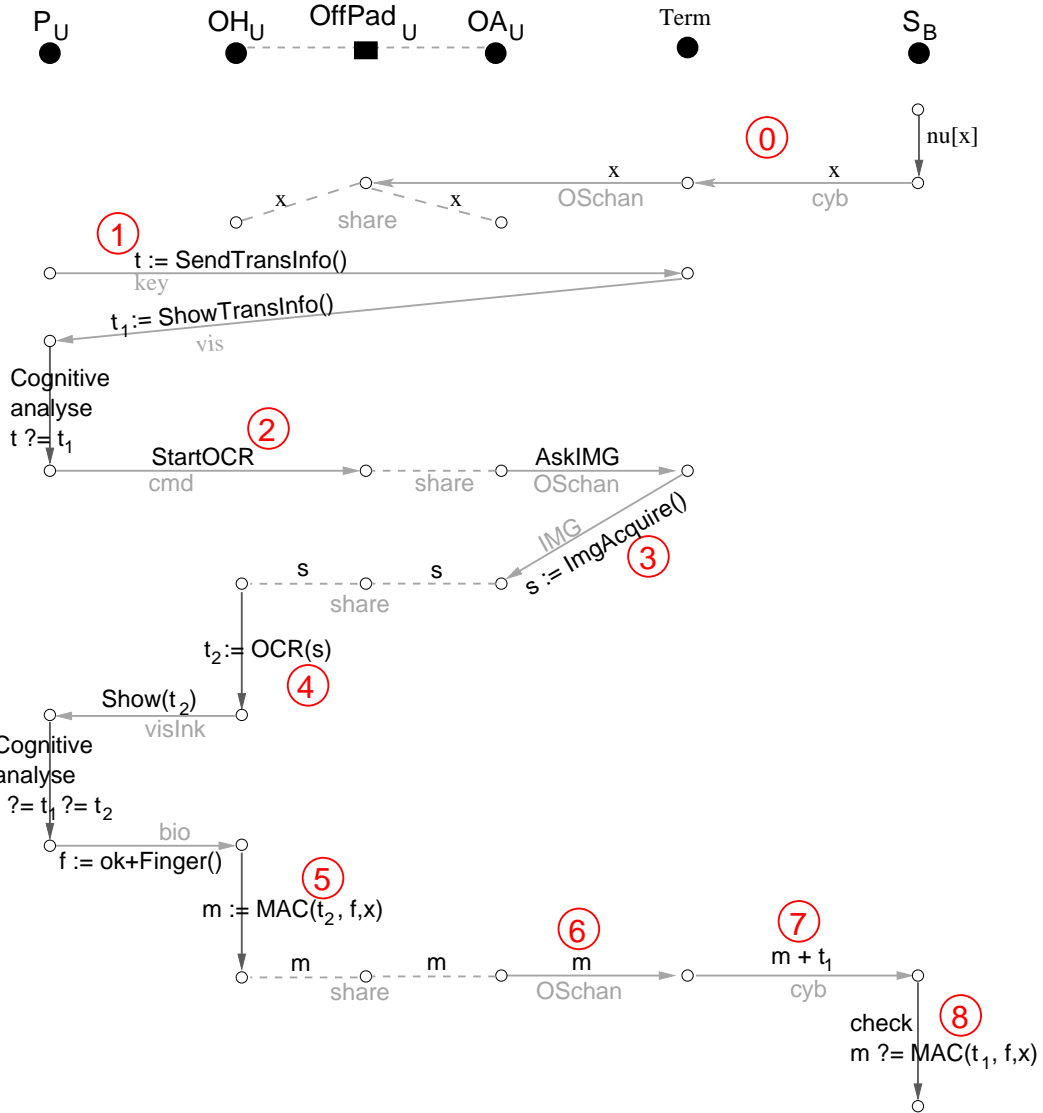


Fig. 3. ANP run for the Data authentication ceremony.

it was displayed to the User) to the Software Agent. This step 3 needs more careful analyses and development.

- The image is shared with the Hardware.
- In the OffPAD Hardware is where the OCR operation takes place (in a trusted environment). This takes as input the acquired image, and outputs some transaction information. For our purposes we can only denote this t_2 differently from before.
- The Transaction information is then Displayed on the InkDisplay to the Persona.
- The Persona can check to see if these transaction information are matching.
- Then the Persona can issues the signing key by providing the Hardware with the Fingerprint. This key can also just be the OffPAD Hardware key (as TazTag specifies in their deliverables when discussing signing of documents using OffPAD). The Fingerprint can mean the Authentication of

the User by the OffPAD Hardware (function discussed by TazTag documents). The Fingerprint can also trigger a command, like “OK, make the MAC and finalize the transaction”.

- Using the key f the OffPAD Hardware produces a MAC (Message Authentication Code) of the transaction information t_2 it extracted from the image; call this m . Note that this could have been done without interaction with the User, but just immediately after the OCR operation, and by using the OffPAD key, since the user has already been authenticated previously to the transaction start. But this is an assumption that we need to be careful about.
- The MAC m is being shared with the Software Agent, which in turn sends it to the Terminal. In the case of the SmartPhone scenario, the Agent communicates through the Operating System message bus. But what exact communication mechanism is abstracted away here, and we only assume to not be so unsecure as a cyber-channel, but maybe under the Operating System.
- The Terminal sends both this message and the transaction information that it wants (here we assume it sends the same transaction that it was showing to the User) to the Server of the Bank through the cyber-channel.
- At the Bank’s Server the MAC is being verified against the transaction information sent by the Terminal.

3.3 API descriptions

By now we can identify several functionalities that need to be provided by the components of the ceremony in order to achieve the run that we just described. We describe these here in the form of API specifications. Later we also describe some sequence diagrams in which these APIs are being used, together with other APIs taken from the documents already provided by TazTag.

1. **DisplayTransInfo()** on the OffPAD Hardware component

purpose: to display the Transaction information on the InkDisplay of the OffPAD Hardware.

This provides trusted *visual* communication with the User.

input: annotated in a specific purpose XML format, which we will call TransXML. The annotations are needed so that we can tell the display how each piece of information to be rendered, like the Bank account larger, or the sum larger.

output: the Status: success or error (what kind of error: parsing, i.e., wrong input, etc.)

2. **OCRTransInfo()** on the OffPAD Hardware component

purpose: to extract from an image the information related to a transaction for a specific Service.

The image needs to satisfy some visual queues so to allow the OCR to recognize easier the relevant information and the respective annotation it should carry.

input: an image (like png, jpeg, etc.) that is provided by the Terminal;

together with a Service Provider ID. We use the ID to tune the extraction process (i.e., apply different parameters or method) depending on the design imposed by the service provider.

output: text in the TransXML format.

3. **MACgen()** on the OffPAD Hardware component

purpose: to generate a MAC for a message (for us containing the transaction information) using the key of the OffPAD hardware which is released by the User through authentication with biometrics. Uses a hash function like SHA-256.

input: the transaction information and a key.

output: a hash of the message.

4. **ShareNFC()** on the OffPAD Hardware and on the OffPAD Software Agent

purpose: to allow for transmission of any kind of data between the two OffPAD components, in both ways.

input: what the NFC standard specifies.

output: same as input.

3.4 A Concrete Application

Let us now consider a concrete implementation of the above ceremony in conjunction with the work of the other partners. See the sequence diagram of Figure 4.

Consider that the Terminal is a SmartPhone with the Android operating system. For the terminal we consider the Browser through which the User interacts by filling in the form sent by the Bank for performing a Transaction.

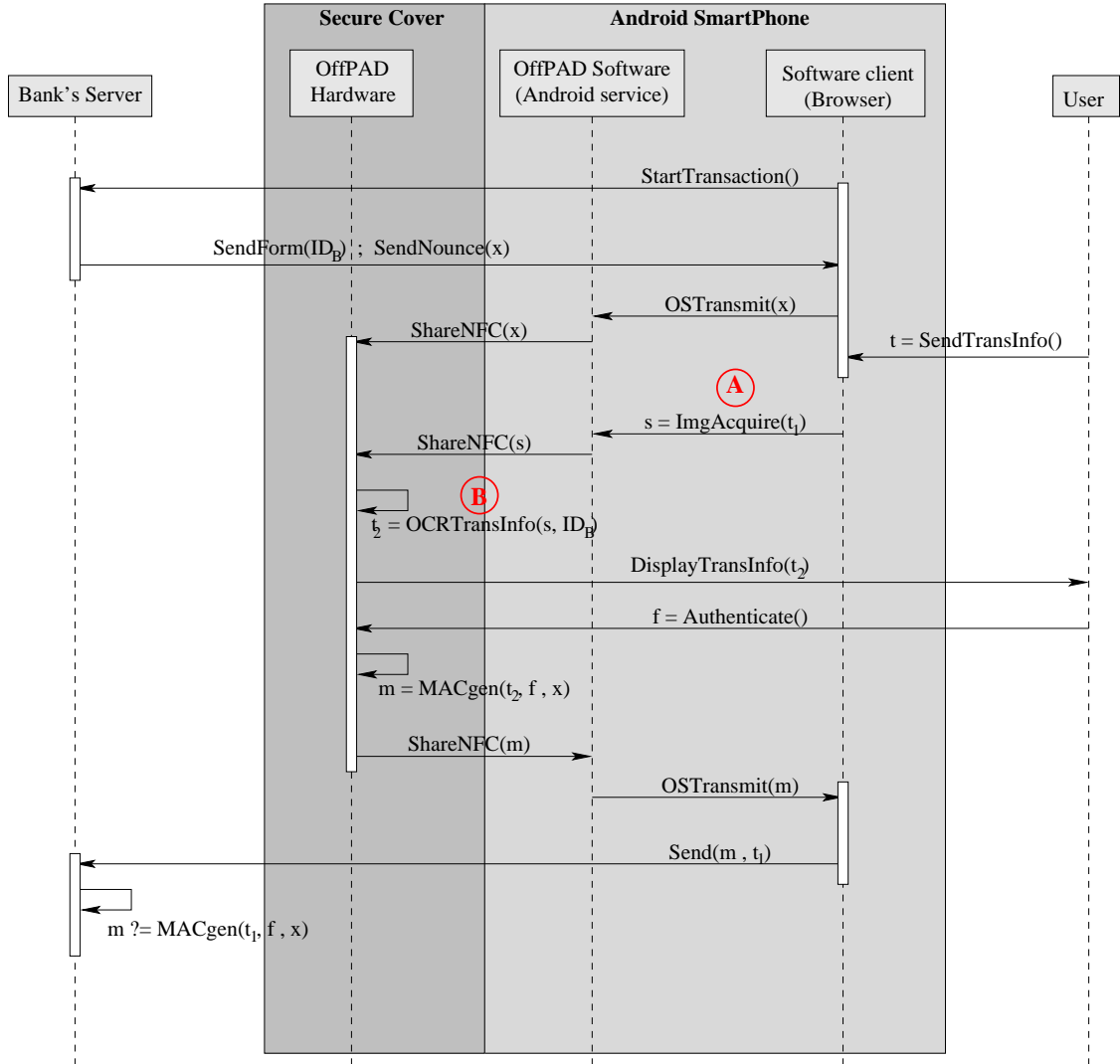


Fig. 4. Sequence diagram for the Data authentication ceremony.

Then consider the OffPAD Agent to be an “Android bound Service”, as described by TazTag in the document about the OffPAD Device Specification. Therefore the Agent runs as an Android app, maybe with different privileges than normal apps. This needs more discussion because it is important to understand the security aspects and what kind of resources are available to the Agent, and in what degree can this access to resources be trusted. For example, can the Agent have access to the camera of the SmartPhone (Terminal); or can the Agent take a screen-shot?

The OffPAD Hardware is attached to the SmartPhone as a back-cover (explained in the documents of TazTag). NFC communication is possible between the back-cover and the SmartPhone.

There are two main aspects of this diagram that need more detailing: these are marked with the red labels A and B. This is where the development mostly takes place.

(A) is where we acquire an image of what the Browser displays to the User, to be used in the Trusted environment.

(B) is the OCR technology implementation, also considering various designs as preferred by the Bank (i.e., is parametrized by the Bank’s identifier).

References

1. Mohammed Alzomai, Bander Alfayyadh, and Audun Jøsang. Display security for online transactions. In *The 5th International Conference for Internet Technology and Secured Transactions (ICITST-2010)*, London, November 2010.
2. Mohammed AlZomai, Bander AlFayyadh, Audun Jøsang, and Adrian McCullag. An Experimental Investigation of the Usability of Transaction Authorization in Online Bank Security Systems. In *The Proceedings of the Australasian Informatin Security Conference (AISC2008)*, Wollongong, Australia, January 2008.
3. Sean Bodmer. SpyEye being kicked to the curb by its customers? Research Note, Damballa Inc. <http://www.damballa.com>, 2012.
4. ITU (CCITT). *Recommendation X.800, Security Architecture for Open Systems Interconnection for CCITT Applications*. International Telecommunications Union (formerly known as the International Telegraph and Telephone Consultative Committee), 1991. (X.800 is a re-edition of IS7498-2).
5. Christian Johansen and Audun Jøsang. Probabilistic Modeling of Humans in Security Ceremonies. In Alessandro Aldini, Fabio Martinelli, and Neeraj Suri, editors, *3rd International Workshop on Quantitative Aspects in Security Assurance (QASA14)*, volume 8872 of *Lecture Notes in Computer Science*, Wroclow, Poland, September 2014. Springer.
6. A. Jøsang, D. Povey, and A. Ho. What You See is Not Always What You Sign. In *Proceedings of the Australian UNIX and Open Systems Users Group Conference (AUUG2002)*, Melbourne, September 2002.
7. K. Kain, S.W. Smith, and R. Asokanm. Digital Signatures and Electronic Documents: A Cautionary Tale. In *Proceedings of IFIP Conference on Communications and Multimedia Security: Advanced Communications and Multimedia Security.*, 2002.
8. Harshit Nayyar. Clash of the Titans: Zeus v SpyEye. SANS Institute InfoSec Reading Room, 2010.
9. Dusko Pavlovic and Catherine Meadows. Actor-network procedures: Modeling multi-factor authentication, device pairing, social interactions. *arXiv.org*, 2011.
10. Cristian Prisacariu. Actor Network Procedures as Psi-calculi for Security Ceremonies. In Sjouke Mauw, Barbara Kordy, and Wolter Pieters, editors, *GraMSec 2014 – International Workshop on Graphical Models for Security*, 2014.
11. Adrian Spalka, Armin B. Cremers, and Hanno Langweg. The fairy tale of ‘What You See Is What You Sign - Trojan Horse Attacks on Software for Digital Signatures. In *IFIP Working Conference on Security and Control of IT in Society-II (SCITS-II)*, Bratislava, Slovakia, June 2001.

12. Arnd Weber. See What You Sign: Secure Implementations of Digital Signatures. In *Proceedings of the International Conference on Intelligence and Services in Networks (ISN 1998)*, pages 509–520, 1998.

Appendix B

GUI Sketches for the Android application



Figure B.1: Application UI sketches - Part I

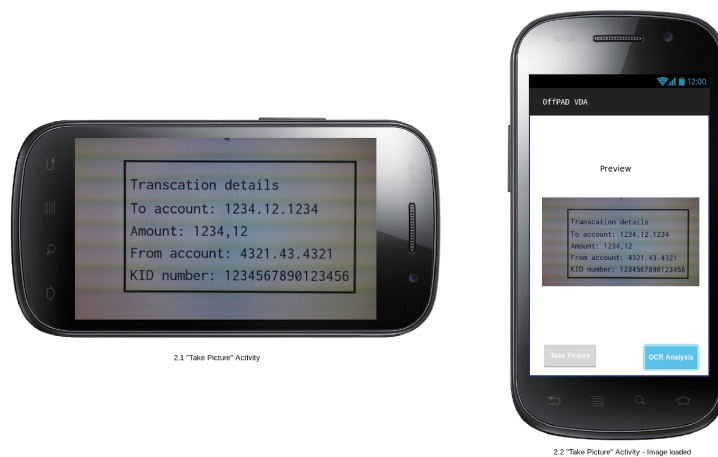


Figure B.2: Application UI sketches - Part II

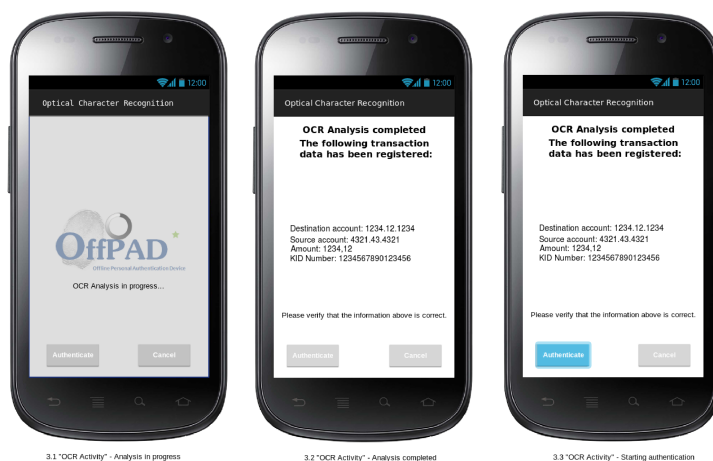


Figure B.3: Application UI sketches - Part III

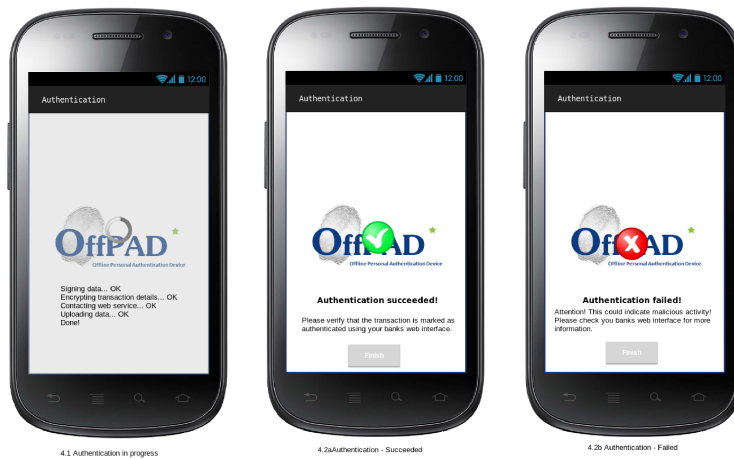


Figure B.4: Application UI sketches - Part IV

Appendix C

Informed Consent Form

Evaluation of e-banking prototype

Informed consent form

Responsible for evaluation:

Marius Portaas Haugen
Department of Informatics
University of Oslo
Telephone: +47 984 47 963
E-mail: mariusp@ifi.uio.no

Motivation

This study seek to evaluate an approach to data authentication in a e-banking use case.

Procedure

If you agree to take part in this study, you will perform a given set of tasks in a test environment, while being observed by an experiment observer. When the tasks are performed, you are asked to answers a questionnaire.

Risk and benefits

There are no identified risks related to take part in this study, nor any benefits.

Your answers will be confidential

The data gathered in this study will be anonymised, and kept completely private. If any analysis of the data is made public, it will not include any data which can identify you.

Participation is voluntary

Taking part in this study is voluntary. You are free to skip any questions, or withdraw from the study completely at any time.

Please sign below to acknowledge that you have read the information above, and that you consent to participate in the study.

Your name: _____

Signature: _____

Location: _____

Date: _____

Appendix D

Task List

OffPAD HCI-Study

Participant Instructions

Introduction

OffPAD is a trusted device to support the different forms of authentication that are necessary for trusted interactions (e.g., user authentication, server authentication and data authentication). One of the features proposed for the OffPAD is data authentication using OCR (Optical Character Recognition).

The objective for this experiment is to evaluate this approach for data authentication in a e-banking scenario. As a participant of this experiment, you are first asked to perform a set of tasks described in the next section. After performing these tasks, you are asked to answer the questions found in the questionnaire attached to this page.

Tasks

In this experiment you will register three e-banking transactions in a online banking system. In front of you is a computer display. In this display, a user interface for a e-banking system is displayed. Next to the keyboard, also located directly in front of you, you find a smart phone providing the OffPAD software.

Transactions to register:

1. Amount: 6199,-
From account: 0429.15.87426
To account: 0429.15.87427
KID: 803113912605
Recipient: "John Doe"
2. Amount: 499,-
From account: 0429.15.87426
To account: 0429.15.87427
KID: None
Recipient: None
3. Amount: 200,-
From account: 0429.15.87426
To account: 0429.15.87427
KID: 071524939825
Recipient: Sammy Smith

Appendix E

Questionnaire

OffPAD HCI-Study

Questionnaire

Please circle the most appropriate selection:

Age: **18-25** **26-35** **36-45** **46-55** **56+**

Gender: **Male** **Female**

1.1

Do you have previous experience with e-banking? **Yes** **No** **Uncertain**

If you responded “No” or “Uncertain” in the previous question, you can skip the questions below, and proceed straight to the questions in in page three (3.1). If you responded “Yes”, please continue answer the questions below.

1.2

What means for user authentication is available in the e-banking system most frequently used by yourself (i.e., how do you log in to the e-banking system)? Circle multiple selections if necessary.

Password **OTP (One time password) sent by SMS** **OTP from code generator**

Other **Uncertain**

1.3

What means for data authentication is available in the e-banking system most frequently used by yourself (i.e., how do you authenticate banking transactions)? Circle multiple selections if necessary.

Password **OTP (One time password sent by SMS** **OTP from code generator**

Smart phone app **Other** **Uncertain**

1.4

BankID is the predominant technology used by Norwegian banks for both user and data authentication. Do you have any experience with BankID? Circle the most appropriate statements, multiple statements if necessary.

1. **I have experience with BankID on Mobile (BankID på mobil)**

2. **I have experience with BankID**

3. **I do not have any experience with BankID**

The following questions is to be answered by those who answered “Yes” in question 1.1 (Do you have previous experience with e-banking?)
For those who answered “No” in question 1.1, please proceed to the questions in the next page.

2.1

Rate agreement or disagreement with the statements below (i.e., check the most appropriate box):

Question	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Making bank transfers using my banks e-banking system is secure					
Making bank transfers using my banks e-banking system is simple					
The approach used to authenticate banking transfers (transactions) in my banks e-banking system is user friendly					
The OffPAD approach for authentication of banking transfers (transactions) is user friendly					
I would consider to use the OffPAD approach when registering bank transfers (transactions), if available in my bank e-banking system					
I would prefer to use the OffPAD approach when registering bank transfers (transactions) over the approach currently provided by my banks e-banking system					
I would consider to use the OffPAD approach when registering bank transfers (transactions) if the OffPAD was implemented as a separate hardware device					
I would consider to use the OffPAD approach when registering bank transfers (transactions), but only if the OffPAD was implemented as an application for my smart phone					

Thank you very much for participating in the testing of the OffPAD data authentication prototype.

The following questions is to be answered by those who answered “No” or “Uncertain” in question 1.1 (Do you have previous experience with e-banking?)

3.1

Rate agreement or disagreement with the statements below (i.e., check the most appropriate box):

Question	Strongly agree	Agree	Neutral	Disagree	Strongly Disagree
Making bank transfers using an e-banking system is secure					
I found it simple to register transactions using the e-banking system evaluated in the study					
If I were to start using a e-banking system for registering transfers (transactions), I would like to use the OffPAD approach					

Thank you very much for participating in the testing of the OffPAD data authentication prototype.

Appendix F

Coding table for questionnaires

OffPAD HCI-Study

Questionnaire – Coding Table

Please circle the most appropriate selection:

Age:	18-25	26-35	36-45	46-55	56+
Code:	1	2	3	4	5
Gender:	Male	Female			
Code:	1	2			

1.1

Do you have previous experience with e-banking?	Yes	No	Uncertain
Code:	1	2	3

If you responded “No” or “Uncertain” in the previous question, you can skip the questions below, and proceed straight to the questions in in page three (3.1). If you responded “Yes”, please continue answer the questions below.

1.2

What means for user authentication is available in the e-banking system most frequently used by yourself (i.e., how do you log in to the e-banking system)? Circle multiple selections if necessary.

Password	OTP (One time password) sent by SMS	OTP from code generator
2	3	5
Other	Uncertain	
7	11	

Unique value=product of every response.

1.3

What means for data authentication is available in the e-banking system most frequently used by yourself (i.e., how do you authenticate banking transactions)? Circle multiple selections if necessary.

Password	OTP (One time password sent by SMS	OTP from code generator
2	3	5
Smart phone app	Other	Uncertain
7	11	13

Unique value=product of every response

1.4

BankID is the predominant technology used by Norwegian banks for both user and data authentication. Do you have any experience with BankID? Circle the most appropriate statements, multiple statements if necessary.

1. **I have experience with BankID on Mobile (BankID på mobil)** – Code: 1

2. **I have experience with BankID** – Code: 2

3. **I do not have any experience with BankID** – Code: 4

Unique value: 1,2,3 or 4

The following questions is to be answered by those who answered “Yes” in question 1.1 (Do you have previous experience with e-banking?)
For those who answered “No” in question 1.1, please proceed to the questions in the next page.

2.1

Rate agreement or disagreement with the statements below (i.e., check the most appropriate box):

Question	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Making bank transfers using my banks e-banking system is secure					
Making bank transfers using my banks e-banking system is simple					
The approach used to authenticate banking transfers (transactions) in my banks e-banking system is user friendly					
The OffPAD approach for authentication of banking transfers (transactions) is user friendly					
I would consider to use the OffPAD approach when registering bank transfers (transactions), if available in my bank e-banking system					
I would prefer to use the OffPAD approach when registering bank transfers (transactions) over the approach currently provided by my banks e-banking system					
I would consider to use the OffPAD approach when registering bank transfers (transactions) if the OffPAD was implemented as a separate hardware device					
I would consider to use the OffPAD approach when registering bank transfers (transactions), but only if the OffPAD was implemented as an application for my smart phone					
Code:	1	2	3	4	5

The following questions is to be answered by those who answered “No” or “Uncertain” in question 1.1 (Do you have previous experience with e-banking?)

3.1

Rate agreement or disagreement with the statements below (i.e., check the most appropriate box):

Question	Strongly agree	Agree	Neutral	Disagree	Strongly Disagree
Making bank transfers using an e-banking system is secure					
I found it simple to register transactions using the e-banking system evaluated in the study					
If I were to start using a e-banking system for registering transfers (transactions), I would like to use the OffPAD approach					
Code:	1	2	3	4	5

Appendix G

Coded data from the questionnaires

Subject	Age	Gender	1.1	1.2	1.3	1.4	2.1A	2.1B	2.1C	2.1D	2.1E	2.1F	2.1G	2.1H	3.1A	3.1B	3.1C
P001	1	2	1	2	3	1	3	1	2	3	4	4	2	2	N/A	N/A	N/A
P002	2	1	1	6	6	3	3	2	5	4	4	5	2	2	N/A	N/A	N/A
P003	1	1	1	7	45	1	1	1	5	5	5	5	2	2	N/A	N/A	N/A
P004	2	1	1	15	35	3	2	4	2	2	3	2	4	4	N/A	N/A	N/A
P005	1	2	1	10	10	2	2	2	2	2	4	4	4	2	N/A	N/A	N/A

Bibliography

- [1] Commerzbank AG. *Safe online banking*. [Online; accessed 2-September-2016]. URL: <https://www.commerzbank.de/portal/en/englisch/products-offers/services/secure-internet-banking/banking.html>.
- [2] Luis von Ahn, Manuel Blum and John Langford. ‘Telling Humans and Computers Apart Automatically’. In: *Commun. ACM* 47.2 (Feb. 2004), pp. 56–60. ISSN: 0001-0782. DOI: 10.1145/966389.966390. URL: <http://doi.acm.org/10.1145/966389.966390>.
- [3] BankID Norge AS. *Oversikt*. [Online; accessed 2-September-2016]. URL: <https://www.bankid.no/bedrift/>.
- [4] Ann Kristin Bentzen. *Dyrt å beholde konto-nummer, men for dyrt?* [Online; accessed 11-August-2016]. URL: <http://www.digi.no/artikler/dyrt-a-beholde-konto-nummer-men-for-dyrt/198807>.
- [5] Daniel Chandler and Rod Munday. *optical character recognition*. URL: <http://www.oxfordreference.com/10.1093/acref/9780199568758.001.0001/acref-9780199568758-e-1931>.
- [6] D. Cooper et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. <http://www.rfc-editor.org/rfc/rfc5280.txt>. RFC Editor, 2008. URL: <http://www.rfc-editor.org/rfc/rfc5280.txt>.
- [7] David Doermann and Karl Tombre. *Handbook of Document Image Processing and Recognition*. Springer Publishing Company, Incorporated, 2014. ISBN: 0857298585, 9780857298584.
- [8] *FRAUD THE FACTS 2016 — THE DEFINITIVE OVERVIEW OF PAYMENT INDUSTRY FRAUD*. [Online; accessed 08-September-2016]. 2016. URL: https://fraudfacts16.financialfraudaction.org.uk/assets/fraud_the_facts.pdf.
- [9] Dieter Gollmann. *Computer Security*. New York, NY, USA: John Wiley & Sons, Inc., 1999. ISBN: 0-471-97844-2.
- [10] Hideaki Goto and Takuma Hoda. ‘Computers Helping People with Special Needs: 14th International Conference, ICCHP 2014, Paris, France, July 9-11, 2014, Proceedings, Part I’. In: ed. by Klaus Miesenberger et al. Cham: Springer International Publishing, 2014. Chap. Real-Time Text Tracking for Text-to-Speech Translation Camera for the Blind, pp. 658–661. ISBN: 978-3-319-08596-8. DOI: 10.

- 1007/978-3-319-08596-8_101. URL: http://dx.doi.org/10.1007/978-3-319-08596-8_101.
- [11] Andy Greenberg. *Hackers Remotely Kill a Jeep on the Highway—With Me in It*. [Online; accessed 22-October-2016]. 2015. URL: <https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/>.
 - [12] Chris Hoffman. *Glossy vs. Matte LCDs: Which Should You Choose When Buying a Display?* [Online; accessed 31-October-2016]. 2014. URL: <http://www.howtogeek.com/181727/glossy-vs.-matte-lcds-which-should-you-choose-when-buying-a-display/>.
 - [13] Google Inc. *Google 2-Step Verification - Features*. [Online; accessed 12-March-2016]. URL: <https://www.google.com/landing/2step/features.html>.
 - [14] E. Kalige and D. Burkey. *A Case Study of Eurograbber: How 36 Million Euros was Stolen via Malware*. [Online; accessed 1-September-2016]. 2012. URL: [https://tweaking.net/files/upload/Eurograbber_White_Paper_281112%20\(2\).pdf](https://tweaking.net/files/upload/Eurograbber_White_Paper_281112%20(2).pdf).
 - [15] Thorsten Kleinjung et al. *Factorization of a 768-bit RSA modulus*. Cryptology ePrint Archive, Report 2010/006. <http://eprint.iacr.org/>. 2010.
 - [16] Kaspersky Labs. *Security Bulletin 2014*. [Online; accessed 29-November-2015]. 2014. URL: <https://securelist.com/files/2014/12/Kaspersky-Security-Bulletin-2014-EN.pdf>.
 - [17] Jonathan Lazar, Jinjuan Heidi Feng and Harry Hochheiser. *Research Methods in Human-Computer Interaction*. Wiley Publishing, 2010. ISBN: 0470723378, 9780470723371.
 - [18] Shujun Li et al. 'Breaking e-Banking CAPTCHAs'. In: *Proceedings of the 26th Annual Computer Security Applications Conference. ACSAC '10*. Austin, Texas, USA: ACM, 2010, pp. 171–180. ISBN: 978-1-4503-0133-6. DOI: 10.1145/1920261.1920288. URL: <http://doi.acm.org/10.1145/1920261.1920288>.
 - [19] Adrian McCullagh and William Caelli. 'Non-repudiation in the digital environment'. In: *First Monday* 5.8 (2000). ISSN: 13960466. URL: <http://firstmonday.org/ojs/index.php/fm/article/view/778>.
 - [20] Alfred J. Menezes, Scott A. Vanstone and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 1996. ISBN: 0849385237.
 - [21] Microsoft. *How to recognize phishing email messages, links, or phone calls*. [Online; accessed 6-March-2016]. URL: <https://www.microsoft.com/en-us/security/online-privacy/phishing-symptoms.aspx>.
 - [22] European Union Agency for Network and Information Security. *Press Release*. [Online; accessed 29-November-2015]. 2012. URL: <https://www.enisa.europa.eu/media/press-releases/eu-cyber-security-agency-enisa-201chigh-roller201d-online-bank-robberies-reveal-security-gaps>.

- [23] Jakob Nielsen. *Quantitative Studies: How Many Users to Test?* [Online; accessed 1-Oktober-2016]. 2006. URL: <https://www.nngroup.com/articles/quantitative-studies-how-many-users/>.
- [24] Jakob Nielsen. *Why You Only Need to Test with 5 Users*. [Online; accessed 5-Oktober-2016]. 2000. URL: <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>.
- [25] Agency for Public Management and eGovernment. *Online public services*. [Online; accessed 11-August-2016]. URL: <http://www.idporten.no/en/online-public-services>.
- [26] Reiner SCT. *Demo chipTAN comfort Reiner SCT*. [Online; accessed 2-September-2016]. URL: http://downloads.reiner-sct.de/flashdemo/tanJack_optic/1024/tanJack_optic.html.
- [27] Helen Sharp, Yvonne Rogers and Jenny Preece. *Interaction Design: Beyond Human Computer Interaction*. John Wiley & Sons, 2007. ISBN: 0470018666.
- [28] R. Smith. 'An Overview of the Tesseract OCR Engine'. In: *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*. ICDAR '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 629–633. ISBN: 0-7695-2822-8. URL: <http://dl.acm.org/citation.cfm?id=1304596.1304846>.
- [29] Ian Sommerville. *Software Engineering*. 9th. USA: Addison-Wesley Publishing Company, 2010. ISBN: 0137035152, 9780137035151.
- [30] SSB. *Bruk av IKT i husholdningene, 2015, 2. kvartal*. [Online; accessed 18-November-2015]. 2015. URL: <http://www.ssb.no/teknologi-og-innovasjon/statistikker/ikthus>.
- [31] USPS. *Postal Mechanization and Early Automation*. [Online; accessed 31-March-2016]. 2012. URL: https://about.usps.com/publications/pub100/pub100_042.htm.
- [32] Shen-Zheng Wang and Hsi-Jian Lee. 'Detection and recognition of license plate characters with different appearances'. In: *Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE*. Vol. 2. 2003, 979–984 vol.2. DOI: 10.1109/ITSC.2003.1252632.
- [33] Lawrence C. Washington and Wade Trappe. *Introduction to Cryptography: With Coding Theory*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2002. ISBN: 0130618144.
- [34] Eric W Weisstein. *Fundamental Theorem of Arithmetic*. [Online; accessed 1-Oktober-2016]. URL: <http://mathworld.wolfram.com/FundamentalTheoremofArithmetic.html>.
- [35] Wikipedia. *Authentication — Wikipedia, The Free Encyclopedia*. [Online; accessed 11-November-2015]. 2015. URL: <https://en.wikipedia.org/wiki/Authentication>.